# 31st Annual European Symposium on Algorithms

**ESA 2023, September 4–6, 2023, Amsterdam, The Netherlands**

Edited by
Inge Li Gørtz
Martin Farach-Colton
Simon J. Puglisi
Grzegorz Herman

*Editors*

**Inge Li Gørtz**
Technical University of Denmark, Lyngby, Denmark
inge@dtu.dk

**Martin Farach-Colton**
Rutgers University, New Brunswick, NJ, USA
martin@farach-colton.com

**Simon J. Puglisi**
University of Helsinki, Finland
simon.j.puglisi@gmail.com

**Grzegorz Herman**
Jagiellonian University, Kraków, Poland
grzegorz.herman@uj.edu.pl

*ACM Classification 2012*
Computing methodologies → Machine learning; Information systems → Point lookups; Mathematics of computing → Algebraic topology; Mathematics of computing → Approximation algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Combinatoric problems; Mathematics of computing → Computations on matrices; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph enumeration; Mathematics of computing → Graphs and surfaces; Mathematics of computing → Graph theory; Mathematics of computing → Matchings and factors; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Random graphs; Mathematics of computing → Stochastic processes; Social and professional topics → Social engineering attacks; Theory of computation → Quantum computation theory; Theory of computation → Algorithm design techniques; Theory of computation → Algorithmic game theory; Theory of computation → Approximation algorithms analysis; Theory of computation → Complexity theory and logic; Theory of computation → Computational complexity and cryptography; Theory of computation → Computational geometry; Theory of computation → Database query processing and optimization (theory); Theory of computation → Data compression; Theory of Computation → Design and analysis of algorithms; Theory of computation → Dynamic graph algorithms; Theory of computation → Facility location and clustering; Theory of computation → Finite Model Theory; Theory of computation → Fixed parameter tractability; Theory of computation → Generating random combinatorial structures; Theory of computation → Graph algorithms analysis; Theory of computation → Logic and verification; Theory of computation → Markov decision processes; Theory of computation → Massively parallel algorithms; Theory of computation → Mathematical optimization; Theory of computation → Mixed discrete-continuous optimization; Theory of computation → Nonconvex optimization; Theory of computation → Online algorithms; Theory of computation → Oracles and decision trees; Theory of computation → Packing and covering problems; Theory of computation → Parallel algorithms; Theory of computation → Parallel computing models; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Pattern matching; Theory of computation → Problems, reductions and completeness; Theory of computation → Quantum complexity theory; Theory of computation → Quantum computation theory; Theory of computation → Quantum query complexity; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Random network models; Theory of computation → Random order and robust communication complexity; Theory of computation → Random projections and metric embeddings; Theory of computation → Routing and network design problems; Theory of computation → Scheduling algorithms; Theory of computation → Semidefinite programming; Theory of computation → Shortest paths; Theory of computation → Sketching and sampling; Theory of computation → Sorting and searching; Theory of computation → Sparsification and spanners; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Theory and algorithms for application domains; Theory of computation → Unsupervised learning and clustering

**ISBN 978-3-95977-295-2**

*Published online and open access by*
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-295-2.

*Publication date*
September, 2023

*Bibliographic information published by the Deutsche Nationalbibliothek*
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.



Digital Object Identifier: 10.4230/LIPIcs.ESA.2023.0

## LIPIcs – Leibniz International Proceedings in Informatics

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



**ISSN 1868-8969**

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

# Contents



31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# Preface

This volume contains the extended abstracts selected for presentation at ESA 2023, the 31st European Symposium on Algorithms. The event was organized by Centrum Wiskunde & Informatica (CWI), Amsterdam, the Netherlands, as a part of ALGO 2023, on September 4–6, 2023.

The scope of ESA includes original, high-quality, theoretical and applied research on algorithms and data structures. Since 2002, it has had two tracks: the Design and Analysis Track (Track A), intended for papers on the design and mathematical analysis of algorithms, and the Engineering and Applications Track (Track B), for submissions that also address real- world applications, engineering, and experimental analysis of algorithms. In 2022, a new track – Track S – was added, inviting contributions that *simplify* algorithmic results. We find that simpler algorithms are easier to implement, bridging the gap between theory and practice, and we find that new simple or elegant proofs are easier to understand and to teach, and may contain interesting new insights whose relevance only the future will reveal.

In response to the call for papers for ESA 2023, 370 papers were submitted, 267 for Track A, 61 for Track B, and 42 for Track S. Paper selection was based on originality, technical quality, exposition quality, and relevance. Each paper received at least three reviews. The program committees selected 103 papers for inclusion in the program: 79 from Track A, 15 from Track B, and 9 for Track S, yielding an overall acceptance rate of about 28%. The presentations of the accepted papers, together with two invited talks by Martin Dietzfelbinger (TU Ilmenau) and Rotem Oshman (Tel Aviv University) promise to make up an exciting program.

The European Association for Theoretical Computer Science (EATCS) sponsored best paper and best student paper awards. A submission was eligible for the best student paper award if all authors were doctoral, master, or bachelor students at the time of submission. For Track A, the best paper award was given to Ursula Hebert-Johnson, Daniel Lokshtanov and Eric Vigoda for the paper "Counting and Sampling Labeled Chordal Graphs in Polynomial Time". For Track B, the best paper award was given to Xiangyun Ding, Xiaojun Dong, Yan Gu, Youzhe Liu and Yihan Sun for the paper "Efficient Parallel Output-Sensitive Edit Distance". The best paper award for Track S was given to Oleg Verbitsky and Maksim Zhukovskii for the paper "Canonization of a Random Graph by Two Matrix-Vector Multiplications". The best student paper award was given to Joakim Blikstad and Peter Kiss for the paper "Incremental (1-$\epsilon$)-Approximate Dynamic Matching in O(poly(1/$\epsilon$)) Update Time".

We wish to thank all the authors who submitted papers for consideration, the invited speakers, the members of the program committees for their hard work, and the over 500 external reviewers who assisted the program committees in the evaluation process. Special thanks go to the organizing committee, who helped us with the organization of the conference.

Information on past ESA symposia, including locations and proceedings, is maintained at `http://esa-symposium.org`.

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# Program Committees

## Track A (Design and Analysis) Program Committee

- Marek Adamczyk, Wroclaw University
- Soheil Behnezhad, Northeastern University
- Mark Bun, Boston University
- Sergio Cabello, University of Ljubljana
- Bhaskar Ray Chaudhury, University of Illinois at Urbana
- Alex Conway, VMWare Research
- Christian Coester, St. Anne's College
- Rathish Das, University of Liverpool
- Anne Driemel, University of Bonn
- Hendrik Fichtenberger, Google Research Zürich
- Moses Ganardi, Max Planck Institute for Software Systems
- Naveen Garg, IIT Delhi
- Cyril Gavoille, LaBRI, University of Bordeaux
- Paweł Gawrychowski, Wroclaw University
- Shay Golan, Reichman and Haifa University
- Inge Li Gørtz, Technical University of Denmark (Chair)
- Carla Groenland, Utrecht University
- Maximilian Probst Gutenberg, ETH Zürich
- Danny Hermelin, Ben-Gurion University
- Giuseppe Italiano, Luiss University
- Lars Jaffke, University of Bergen
- Haim Kaplan, Tel Aviv University
- William Kuszmaul, MIT
- Frédéric Magniez, CNRS, IRIF
- Miguel Mosteiro, Pace University
- Yasamin Nazari, University of Salzburg and VU Amsterdam
- Eunjin Oh, Pohang University of Science and Technology
- Merav Parter, Weizmann Institute of Science
- Adam Polak, Max Planck Institute for Informatics
- Kent Quanrud, Purdue University
- Nicola Prezza, Ca' Foscari University of Venice
- Jakub Radoszewski, University of Warsaw and Samsung R&D Warsaw
- Malin Rau, Hamburg University
- Liam Roditty, Bar-Ilan
- Marc Roth, University of Oxford
- Saeed Seddighin, Toyota Technological Institute at Chicago
- Francesco Silvestri, University of Padova
- Christian Sohler, University of Cologne
- Tatiana Starikovskaya, École Normale Supérieure Paris
- Jukka Suomela, Aalto University
- Alexandru Tomescu, University of Helsinki
- Meng-Tsung Tsai, Academia Sinica
- Ivor Djinn Van Der Hoog, Technical University of Denmark

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- Nicole Wein, DIMACS
- Karol Węgrzycki, Max Planck Institute for Informatics
- Anna Zych-Pawlewicz, University of Warsaw

## Track B (Engineering and Applications) Program Committee

- Jarno Alanko, University of Helsinki
- Giulia Bernardini, University of Trieste
- Vincenzo Bonifaci, University of Rome Tre
- Katrin Casel, Humboldt University of Berlin
- David Coudert, INRIA Sophia Antipolis
- Donatella Firmani, University of Rome, La Sapienza
- Klaus Jansen, Christian-Albrechts University of Kiel
- Quanquan C. Liu, Northwestern University
- Tamara Mchedlidze, University of Utrecht
- Prashant Pandey, University of Utah
- Giulio Piribi, Ca' Foscari University of Venice
- Simon J. Puglisi, University of Helsinki (Chair)
- Christian Schulz, University of Heidelberg
- Sabine Storandt, University of Konstanz
- David Tench, Rutgers University
- Helen Xu, Lawrence Berkeley National Lab

## Track S (Simplicity) Program Committee

- Sepehr Assadi, Rutgers University and University of Waterloo
- Deeparnab Chakrabarty, Dartmouth College
- Graham Cormode, University of Warwick
- Leah Epstein, University of Haifa
- Martin Farach-Colton, Rutgers University (Chair)
- Magnus M. Halldorsson, Reykjavik University
- John Iacono, Université libre de Bruxelles
- Dominik Kempa, Stony Brook University
- John Lapinskas, University of Bristol
- Kasper Green Larsen, Aarhus University
- Michał Pilipczuk, University of Warsaw
- Aditya Potukuchi, York University
- Ronitt Rubinfeld, MIT and Tel-Aviv University
- Chris Schwiegelshohn, Aarhus University
- Vera Traub, University of Bonn
- Przemek Uznanski, University of Wroclaw
- Oren Weimann, University of Haifa
- Omri Weinstein, Hebrew University

# List of External Reviewers

Mikkel Abrahamsen
Peyman Afshani
Akanksha Agrawal
Kunal Agrawal
Jungho Ahn
Hugo Akitaya
Shyan Akmal
Hannaneh Akrami
Sharareh Alipour
Saeed Akhoondian Amiri
Ashwani Anand
Nima Anari
Robert Andrews
Spyros Angelopoulos
Shinwoo An
Antonios Antoniadis
Elena Arseneva
Vikraman Arvind
Amir Azarmehr
Arturs Backurs
Narmina Baghirova
Mohammad Reza Bahrami
Lorenzo Balzotti
Jérémy Barbay
Surender Baswana
Gabriel Bathie
Pascal Baumann
Ruben Becker
Aleksandrs Belovs
Omri Ben-Eliezer
Matthias Bentert
Lorenzo Beretta
Nils Van de Berg
Benjamin Bergougnoux
Sarita de Berg
Sebastian Berndt
Aaron Bernstein
Ivona Bezakova
Subhash Bhagat
Arghya Bhattacharya
Sriram Bhyravarapu
Elena Biagi
Therese Biedl
Davide Bilò
Amartya Shankha Biswas
Mitchell Black
Jaroslaw Blasiok
Thomas Bläsius
Joakim Blikstad
Ivan Bliznets
Johannes Blum
Manuel Bodirsky
Hans L. Bodlaender
Greg Bodwin
Martin Böhm
Itai Boneh
Fabrizio Boninsegna
Édouard Bonnet
Flavia Bonomo
Yvan Le Borgne
Prosenjit Bose
Nicolas Bousquet
Joshua Brakensiek
Cornelius Brand
Adrián Gómez Brandón
Sebastian Brandt
Karl Bringmann
Hauke Brinkop
Reilly Browne
Frederik Brüning
Austin Buchanan
Niv Buchbinder
Moritz Buchem
Maike Buchin
Laurent Bulteau
Manuel Cáceres
Daniele Carnevale
Arielle Carr
Matteo Ceccarello
Davide Cenzato
Hsien-Chih Chang
Yi-Jun Chang
T-H. Hubert Chan
Timothy M. Chan
Vaggos Chatziafratis
Chandra Chekuri
Ho-Lin Chen
Li Chen

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Yuvaraj Chesetti
Sinho Chewi
Ashish Chiplunkar
Rajesh Chitnis
Janka Chlebikova
Kyungjin Cho
Keerti Choudhary
Aleksander Bjoern Grodt Christiansen
Ilan Cohen
Alessio Conte
Arjan Cornelissen
Sabine Cornelsen
Alex Crane
Andrés Cristi
Artur Czumaj
Rajni Dabas
Konrad K. Dabrowski
Mina Dalirrooyfard
Justin Dallant
Clément Dallard
Debarati Das
Daniel DeLayo
Holger Dell
Shichuan Deng
Max Deppert
Antoine Deza
Diego Diaz
Michael Dinitz
Yann Disser
Xiaojun Dong
Dean Doron
Michal Dory
David Doty
Andrew Draganov
Pål Grønås Drange
Jan Dreier
Matthew Drescher
Lukas Drexler
François Dross
Aditi Dudeja
Bartlomiej Dudek
Christoph Dürr
Zdenek Dvorak
Matthijs Ebbens
Franziska Eberle
Eduard Eiben
Antoine El-Hayek
Jonas Ellert
Matthias Englert
David Eppstein
Massimo Equi
Thomas Erlebach
Guy Even
Piotr Faliszewski
Austen Z. Fan
Marcelo Fonseca Faraj
Alireza Farhadi
Lene Favrholdt
Mojtaba Fayaz-Bakhsh
Sándor Fekete
Moran Feldman
Andreas Emil Feldmann
Daniele Ferone
Yuval Filmus
Arnold Filtser
Nick Fischer
Nick Fisher
Maxime Flin
Jacob Focke
Sebastian Forster
Dvir Fried
Nicole Funk
Eric Fusy
Esther Galby
Waldo Gálvez
Ankit Garg
Andrea Gasparin
Serge Gaspers
Surya Teja Gavva
Loukas Georgiadis
Rong Ge
Samah Ghazawi
Daniel Gibney
Jacob Gilbert
Gilvir Gill
Lars Gottesbüren
Themistoklis Gouleakis
Garance Gourdel
Fabrizio Grandoni
Catherine Greenhill
Andreas Grigorjew
David Gross
Ernestine Großmann
Christoph Grunau
Frédéric Guinand
Maximilian Hahn-Klimroth

Thekla Hamm
Yassine Hamoudi
Kathrin Hanauer
Sariel Har-Peled
Juha Harviainen
Niko Hastrich
Klaus Heeger
Annika Hennes
John Hershberger
D. Ellis Hershkowitz
Tobias Heuer
Yuya Higashikawa
Jan Höckendorff
Ruben Hoeksma
Felix Hommelsheim
Wing-Kai Hon
Gary Hoppenworth
Tao Hou
Shang-En Huang
Yushen Huang
Thore Husfeldt
Edin Husic
Dylan Hyatt-Denesik
Tanmay Inamdar
Davis Issac
Yuval Itzhaki
Yuni Iwamasa
Hugo Jacob
Shayan Chashm Jahan
Rhea Jain
Wojciech Janczewski
Seyed Mohammad Seyed Javadi
Rajesh Jayaram
Iwan Jensen
Łukasz Jeż
Shaofeng Jiang
Shunhua Jiang
Zhihao Jiang
Ce Jin
Rob Johnson
Seungbum Jo
Kai Kahler
Shahin Kamali
Naoyuki Kamiyama
Lior Kamma
Shunsuke Kanda
Adam Karczmarz
Matti Karppa
Manas Jyoti Kashyop
Telikepalli Kavitha
Bart de Keijzer
Leon Kellerhals
Mehran Khodabandeh
Eun Jung Kim
Sung-Hwan Kim
Peter Kiss
Linda Kleist
Fabian Klute
Marina Knittel
Yusuke Kobayashi
Laura Vargas Koch
Bojana Kodric
Benedikt Kolbe
Hanna Komlos
Athanasios Konstantinidis
Dominik Köppl
Ama Koranteng
Tuukka Korhonen
Irina Kostitsyna
Laszlo Kozma
Stefan Kratsch
Nils Kriege
Ravishankar Krishnaswamy
Amer Krivosija
Myroslav Kryven
Ariel Kulik
Pooja Kulkarni
Rucha Kulkarni
Nikhil Kumar
Pascal Kunz
David Kutner
Oded Lachish
Jakub Łącki
Bundit Laekhanukit
Elmar Langetepe
Matthias Lanzinger
Alexandra Lassota
Christian Janos Lebeda
Tuomo Lehtilä
Hung Le
Johannes Lengler
Jérôme Leroux
Stefano Leucci
Asaf Levin
Roie Levin
Jason Li

# List of External Reviewers

Lawrence Li
Paloma de Lima
Antoine Limasset
Alexander Lindermayr
Shaohua Li
Yang Liu
Vasilis Livanos
William Lochet
Yaowei Long
Alexandre Louvet
Anna Lubiw
Aleksander Łukasiewicz
Marten Maack
Konrad Majewski
Konstantin Makarychev
Frederik Mallmann-Trenn
Nikhil Mande
Bodo Manthey
Pasin Manurangsi
Dániel Marx
Weiyun Ma
Will Ma
Hunter McCoy
Ciaran McCreesh
Tamara Mchedlidze
Simon Meierhans
Lucas Meijer
Arturo Merino
George Mertzios
Martin Milanič
Victor Milenkovic
Till Miltzow
Pranabendu Misra
Joseph Mitchell
Parth Mittal
Matthias Mnich
Hendrik Molter
Tobias Mömke
Morteza Monemizadeh
David Mount
Ramin Mousavi
Marcin Mucha
Moritz Muehlenthaler
Anish Mukherjee
Wolfgang Mulzer
Alexander Munteanu
Harihara Subrahmaniam Muralidharan
Wojciech Nadara
Yuto Nakashima
Vasileios Nakos
Giacomo Nannicini
Seffi Naor
Shyam Narayanan
Anurag Murty Naredla
Bento Natura
Jesper Nederlof
Yakov Nekrich
Daniel Neuen
Stefan Neumann
Alantha Newman
Nicolas Nisse
Martin Nöllenburg
André Nusser
Pranav Nuti
Pierre Ohlmann
Seunghyeok Oh
Yoshio Okamoto
Keisuke Okumura
Jan Olkowski
Shmuel Onn
Tim Oosterwijk
Tim Ophelders
Giacomo Ortali
George Osipov
Sang-il Oum
Shreyas Pai
Katarzyna Paluch
Sukanya Pandey
Fahad Panolan
Irene Parada
Nikos Parotsidis
Galina Pass
Pan Peng
Fedor Petrov
Asaf Petruschka
Jeff Phillips
Gloria Pietropolli
Marcin Pilipczuk
Karol Pokorski
Guido Tagliavini Ponce
Aleksandr Popov
Alex Pothen
Wojciech Przybyszewski
Ioannis Psarros
Nidhi Purohit
Edward Pyne

Benjamin Qi
Dror Rawitz
Meghana M Reddy
Rebecca Reiffenhäuser
Janina Reuter
Rojin Rezvan
Nicola Rizzo
Heiko Röglin
Dennis Rohde
Lars Rohwedder
Jérémie Roland
Massimiliano Rossi
Ignaz Rutter
Sushant Sachdeva
Lakshay Saggi
Vibha Sahlot
Hamed Saleh
Luca Saluzzi
Mohammad Saneian
Richard Santiago
Thatchaphol Saranurak
Ilie Sarpe
Jonas Sauer
Saket Saurabh
Gaia Saveri
Philipp Schepper
Kevin Schewior
Martin Schirneck
Jens Schlöter
Melanie Schmidt
Sebastian Schmidt
Lia Schütze
Luca Pepè Sciarria
Masood Seddighin
Daniel Seemaier
François Sellier
Rik Sengupta
Sayantan Sen
Dvir Shabtay
Hadas Shachnai
Golnoosh Shahkarami
Vihan Shah
Eklavya Sharma
Roohani Sharma
Moshe Shechner
Ruoqi Shen
Zheqi Shen
Benwei Shi
Yoshihiro Shibuya
Jessica Shi
Jihun Shin
Sebastian Siebertz
Rodrigo Silveira
Bertrand Simon
Kirill Simonov
Shikha Singh
Sahil Singla
Nodari Sitchinava
Antonis Skarlatos
Karthik C. S.
Piotr Skowron
Friedrich Slivovsky
Lucas Slot
Michiel Smid
Dina Sokol
Marek Sokołowski
Shay Solomon
Frank Sommer
Gregory Sorkin
Sophie Spirkl
Joachim Spoerhase
Ramanujan M. Sridharan
Frank Staals
Jack Stade
Lorenzo De Stefani
Jens Stoye
Carmen Strassle
Ondrej Suchy
Janani Sundaresan
Hari Sundar
Yihan Sun
Krister Swenson
Dániel Szabó
Krisztina Szilagyi
Mehrdad Tahvilian
Wai Ming Tai
Prafullkumar Tale
Nimrod Talmon
Zhihao Gavin Tang
Zihan Tan
Sharma V. Thankachan
Thomas Thierauf
Théophile Thiery
Mads Toftrup
Yu Tong
Szymon Toruńczyk

Csaba Toth
Ohad Trabelsi
Alok Tripathy
Abner Turkieltaub
Malte Tutas
Ta-Wei Tu
Arash Vaezi
Kasturi Varadarajan
Arsen Vasilyan
Federico Julian Camerota Verdù
Laurent Viennot
Tijn de Vos
Hoa Vu
Magnus Wahlström
David Wajc
Bartosz Walczak
Tomasz Walen
Jose L. Walteros
Chen Wang
Daochen Wang
Hung-Lung Wang
Weihang Wang
Yanheng Wang
Yipu Wang
Yuyan Wang
Justin Ward
Omer Wasim
Philip Wellnitz
Leo Wennmann
Evan West
Brian Wheatman
Sebastian Wiederrecht
Andreas Wiese
Sebastian Wild
Michal Wlodarczyk
Sampson Wong
Kaiyu Wu
Ning Xie
Chao Xu
Jie Xue
Yinzhan Xu
Zhou Xu
Mingquan Ye
Hsu-Chun Yen
Fang-Yi Yu
Viktor Zamaraev
Or Zamir
Hongyang Zhang
Peng Zhang
Tianyi Zhang
Xinzhi Zhang
Zhiwei Zhang
Yiming Zhao
Da Wei Zheng
Hong Zhou
Rudy Zhou
Samson Zhou
Yimin Zhu
Isabella Ziccardi
Wiktor Zuba
Michael Zündorf
Goran Zuzic

# On Hashing by (Random) Equations

**Martin Dietzfelbinger**
Technische Universität Ilmenau, Germany

## Abstract

The talk will consider aspects of the following setup: Assume for each (key) $x$ from a set $\mathcal{U}$ (the universe) a vector $a_x = (a_{x,0}, \ldots, a_{x,m-1})$ has been chosen. Given a list $Z = (z_i)_{i\in[m]}$ of vectors in $\{0,1\}^r$ we obtain a mapping

$$\varphi_Z : \mathcal{U} \to \{0,1\}^r, x \mapsto \langle a_x, Z\rangle := \bigoplus_{i\in[m]} a_{x,i} \cdot z_i,$$

where $\bigoplus$ is bitwise XOR. The simplest way for creating a data structure for calculating $\varphi_Z$ is to store $Z$ in an array $Z[0..m-1]$ and answer a query for $x$ by returning $\langle a_x, Z\rangle$. The length $m$ of the array should be $(1+\varepsilon)n$ for some small $\varepsilon$, and calculating this inner product should be fast. In the focus of the talk is the case where for all or for most of the sets $S \subseteq \mathcal{U}$ of a certain size $n$ the vectors $a_x, x \in S$, are linearly independent. Choosing $Z$ at random will lead to hash families of various degrees of independence. We will be mostly interested in the case where $a_x, x \in \mathcal{U}$ are chosen independently at random from $\{0,1\}^m$, according to some distribution $\mathcal{D}$. We wish to construct (static) *retrieval data structures*, which means that $S \subset \mathcal{U}$ and some mapping $f : S \to \{0,1\}^r$ are given, and the task is to find $Z$ such that the restriction of $\varphi_Z$ to $S$ is $f$. For creating such a data structure it is necessary to solve the linear system

$$(a_x)_{x\in S} \cdot Z = (f(x))_{x\in S}$$

for $Z$. Two problems are central: Under what conditions on $m$ and $\mathcal{D}$ can we expect the vectors $a_x, x \in S$ to be linearly independent, and how can we arrange things so that in this case the system can be solved fast, in particular in time close to linear (in $n$, assuming a reasonable machine model)? Solutions to these problems, some classical and others that have emerged only in recent years, will be discussed.

**2012 ACM Subject Classification** Theory of computation → Sorting and searching; Theory of computation → Randomness, geometry and discrete structures

**Keywords and phrases** Hashing, Retrieval, Linear equations, Randomness

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.1

**Category** Invited Talk

# On Diameter Approximation in Directed Graphs

**Amir Abboud**
Weizmann Institute of Science, Rehovot, Israel

**Mina Dalirrooyfard**
Massachusetts Institute of Technology, Cambridge, MA, USA

**Ray Li**
University of California Berkeley, CA, USA

**Virginia Vassilevska Williams**
Massachusetts Institute of Technology, Cambridge, MA, USA

## Abstract

Computing the diameter of a graph, i.e. the largest distance, is a fundamental problem that is central in fine-grained complexity. In undirected graphs, the Strong Exponential Time Hypothesis (SETH) yields a lower bound on the time vs. approximation trade-off that is quite close to the upper bounds.

In *directed* graphs, however, where only some of the upper bounds apply, much larger gaps remain. Since $d(u, v)$ may not be the same as $d(v, u)$, there are multiple ways to define the problem, the two most natural being the *(one-way) diameter* ($\max_{(u,v)} d(u, v)$) and the *roundtrip diameter* ($\max_{u,v} d(u, v) + d(v, u)$). In this paper we make progress on the outstanding open question for each of them.

- We design the first algorithm for diameter in sparse directed graphs to achieve $n^{1.5-\varepsilon}$ time with an approximation factor better than 2. The new upper bound trade-off makes the directed case appear more similar to the undirected case. Notably, this is the first algorithm for diameter in sparse graphs that benefits from fast matrix multiplication.
- We design new hardness reductions separating roundtrip diameter from directed and undirected diameter. In particular, a 1.5-approximation in subquadratic time would refute the All-Nodes $k$-Cycle hypothesis, and any $(2 - \varepsilon)$-approximation would imply a breakthrough algorithm for approximate $\ell_\infty$-Closest-Pair. Notably, these are the first conditional lower bounds for diameter that are not based on SETH.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness; Theory of computation → Graph algorithms analysis

**Keywords and phrases** Diameter, Directed Graphs, Approximation Algorithms, Fine-grained complexity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.2

**Related Version** *Full Version*: https://arxiv.org/abs/2307.07583

**Funding** *Amir Abboud*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No 101078482). Additionally, Amir Abboud is supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.
*Mina Dalirrooyfard*: Partially supported by an Akamai Fellowship.
*Ray Li*: Supported by the NSF Mathematical Sciences Postdoctoral Research Fellowships Program under Grant DMS-2203067, and a UC Berkeley Initiative for Computational Transformation award.
*Virginia Vassilevska Williams*: Partially supported by the National Science Foundation Grant CCF-2129139.

**Acknowledgements** We would like to thank Piotr Indyk, Karthik C.S., and the participants of the Fine-Grained Approximation Algorithms & Complexity Workshop (FG-APX 2019) at Bertinoro 2019 for many helpful discussions.





# 1 Introduction

The diameter of the graph is the largest shortest paths distance. A very well-studied parameter with many practical applications (e.g. [23, 36, 45, 15]), its computation and approximation are also among the most interesting problems in Fine-Grained Complexity (FGC). Much effort has gone into understanding the approximation vs. running time tradeoff for this problem (see the survey [43] and the progress after it [14, 13, 34, 35, 28, 25]).

Throughout this introduction we will consider $n$-vertex and $m$-edge graphs that, for simplicity, are *unweighted* and *sparse* with $m = n^{1+o(1)}$ edges[1]. The diameter is easily computable in $\tilde{O}(mn) = n^{2+o(1)}$ time[2] by computing All-Pairs Shortest Paths (APSP). One of the first and simplest results in FGC [41, 46] is that any $O(n^{2-\varepsilon})$ time algorithm for $\varepsilon > 0$ for the exact computation of the diameter would refute the well-established Strong Exponential Time Hypothesis (SETH) [30, 18]. Substantial progress has been achieved in the last several years [41, 19, 14, 13, 34, 35, 28, 25], culminating in an approximation/running time lower bound tradeoff based on SETH, showing that even for undirected sparse graphs, for every $k \geq 2$, there is no $2 - 1/k - \delta$-approximation algorithm running in $\tilde{O}(n^{1+1/(k-1)-\varepsilon})$ time for some $\delta, \varepsilon > 0$.

In terms of upper bounds, the following three algorithms work for both undirected and directed graphs:

1. compute APSP and take the maximum distance, giving an exact answer in $\tilde{O}(n^2)$ time,
2. compute single-source shortest paths from/to an arbitrary node and return the largest distance found, giving a 2-approximation in $\tilde{O}(n)$ time, and
3. an algorithm by [41, 19] giving a 3/2-approximation in $\tilde{O}(n^{1.5})$ time.

For undirected graphs, there are some additional algorithms, given by Cairo, Grossi and Rizzi [17] that qualitatively (but not quantitatively) match the tradeoff suggested by the lower bounds: for every $k \geq 1$ they obtain an $\tilde{O}(n^{1+1/(k+1)})$ time, almost-$(2 - 1/2^k)$ approximation algorithm, meaning that there is also a small constant additive error.

The upper and lower bound tradeoffs for undirected graphs are depicted in Figure 1 ; a gap remains (depicted as white space) because the two trade-offs have different rates. In directed graphs, however, the gap is significantly larger because an upper bound trade-off is missing (the lower bound tradeoff follows immediately because it is a harder problem). One could envision for instance, that the conditional lower bounds for directed diameter could be strengthened to show that if one wants a $(2 - \varepsilon)$-approximation algorithm, then it must take at least $n^{1.5-o(1)}$ time. Since the work of [17], the main open question (also asked by [43]) for diameter algorithms in directed graphs has been:

*Why are there only three approximation algorithms for directed diameter, but undirected diameter has an infinite approximation scheme? Is directed diameter truly harder, or can one devise further approximation algorithms for it?*

[1] Notably, however, our algorithmic results hold for general graphs, and our hardness results hold even for very sparse graphs.

[2] The notation $\tilde{O}(f(n))$ denotes $O(f(n) \operatorname{poly} \log(f(n)))$.

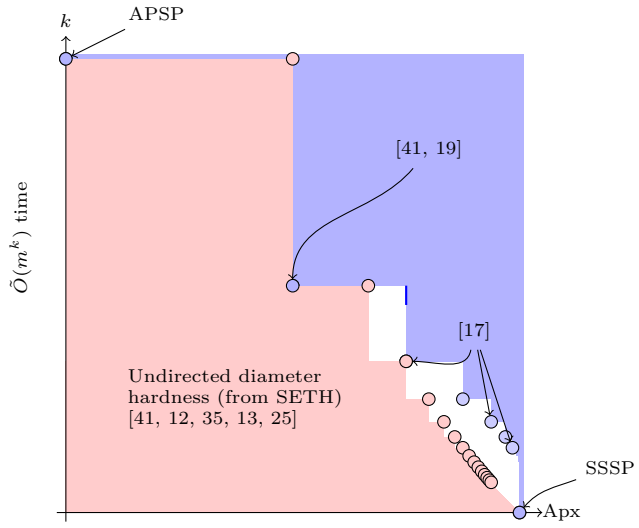



**Figure 1** Undirected diameter algorithms and hardness.

## Directed is Closer to Undirected

Our first result is that one *can* devise algorithms for directed diameter with truly faster running times than $n^{1.5}$, and approximation ratios between 3/2 and 2. It turns out that the directed case has an upper bound tradeoff as well, albeit with a worse rate than in the undirected case. Conceptually, this brings undirected and directed diameter closer together. See Figure 2 for our new algorithms.

▶ **Theorem 1.** *Let $k = 2^{t+2}$ for a nonnegative integer $t \geq 0$. For every $\varepsilon > 0$ (possibly depending on $m$), there exists a randomized $2-\frac{1}{k}+\varepsilon$-approximation algorithm for the diameter of a directed weighted graphs in time $\tilde{O}(m^{1+\alpha}/\varepsilon)$, for*

$$\alpha = \frac{2(\frac{2}{\omega-1})^t - \frac{(\omega-1)^2}{2}}{(\frac{2}{\omega-1})^t(7-\omega) - \frac{\omega^2-1}{2}}.$$

The constant $2 \leq \omega < 2.37286$ in the theorem refers to the fast matrix multiplication exponent [6]. A surprising feature of our algorithms is that we utilize fast matrix multiplication techniques to obtain faster algorithms for a problem in sparse graphs. Prior work on shortest paths has often used fast matrix multiplication to speed-up computations, but to our knowledge, all of this work is for dense graphs (e.g. [7, 44, 47, 24]). Breaking the $n^{1.5}$ bound with a combinatorial algorithm is left as an open problem.

## Roundtrip is Harder

One unsatisfactory property of the shortest paths distance measure in directed graphs is that it is not symmetric ($d(u,v) \neq d(v,u)$) and is hence not a metric. Another popular distance measure used in directed graphs that is a metric is the roundtrip measure. Here the *roundtrip distance* $\tilde{d}(u,v)$ between vertices $u, v$ is $d(u,v) + d(v,u)$.

Roundtrip distances were first studied in the distributed computing community in the 1990s [22]. In recent years, powerful techniques were developed to handle the fast computation of sparse roundtrip spanners, and approximations of the *minimum* roundtrip distance, i.e.

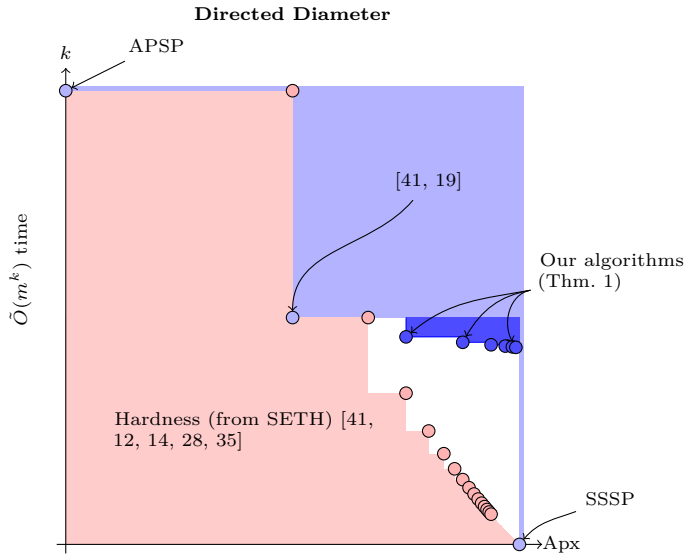



**Figure 2** Directed diameter algorithms and hardness. All tradeoffs hold for both weighted and unweighted graphs (though citations may differ for weighted vs. unweighted).

the shortest cycle length, the girth, of a directed graph [38, 21, 26, 20]. These techniques give hope for new algorithms for the *maximum* roundtrip distance, the roundtrip diameter of a directed graph.

Only the first *two* algorithms in the list in the beginning of the introduction work for roundtrip diameter: compute an exact answer by computing APSP, and a linear time 2-approximation that runs SSSP from/to an arbitrary node. These two algorithms work for any distance *metric*, and surprisingly there have been no other algorithms developed for roundtrip diameter. The only fine-grained lower bounds for the problem are the ones that follow from the known lower bounds for diameter in undirected graphs, and these cannot explain why there are no known subquadratic time algorithms that achieve a better than 2-approximation.

*Are there $O(n^{2-\varepsilon})$ time algorithms for roundtrip diameter in sparse graphs that achieve a $2-\delta$-approximation for constants $\varepsilon, \delta > 0$?*

This question was considered e.g. by [4] who were able to obtain a hardness result for the related roundtrip radius problem, showing that under a popular hypothesis, such an algorithm for roundtrip radius does not exist. One of the main questions studied at the "Fine-Grained Approximation Algorithms and Complexity Workshop" at Bertinoro in 2019 was to obtain new algorithms or hardness results for roundtrip diameter. Unfortunately, however, no significant progress was made, on either front.

The main approach to obtaining hardness for roundtrip diameter, was to start from the Orthogonal Vectors (OV) problem and reduce it to a gap version of roundtrip diameter, similar to all known reductions to (other kinds of) diameter approximation hardness. Unfortunately, it has been difficult to obtain a reduction from OV to roundtrip diameter that has a larger gap than that for undirected diameter; in Section 4.1 we give some intuition for why this is the case.

In this paper we circumvent the difficulty by giving stronger hardness results for roundtrip diameter starting from *different* problems and hardness hypotheses. We find this intriguing because all previous conditional lower bounds for (all variants of) the diameter problem were

based on SETH. In particular, it gives a new approach for resolving the remaining gaps in the undirected case, where higher SETH-based lower bounds are provably impossible (under the so-called NSETH) [35].

Our first negative result conditionally proves that any $5/3-\varepsilon$ approximation for roundtrip requires $n^{2-o(1)}$ time; separating it from the undirected and the directed one-way cases where a 1.5-approximation in $\tilde{O}(n^{1.5})$ time is possible. This result is based on a reduction from the so-called All-Nodes $k$-Cycle problem.

▶ **Definition 2** (All-Nodes $k$-Cycle in Directed Graphs). *Given a $k$ partite directed graph $G=(V,E), V=V_1\cup\cdots\cup V_k$, whose edges go only between "adjacent" parts $E\subseteq\bigcup_{i=1}^{k} V_i\times V_{i+1 \mod k}$, decide if all nodes $v\in V_1$ are contained in a $k$-cycle in $G$.*

This problem can be solved for all $k$ in time $O(nm)$, e.g. by running an APSP algorithm, and in subquadratic $O(m^{2-1/k})$ for any fixed $k$ [8]. Breaking the quadratic barrier for super-constant $k$ has been a longstanding open question; we hypothesize that it is impossible.

▶ **Hypothesis 3.** *No algorithm can solve the All-Nodes $k$-Cycle problem in sparse directed graphs for all $k\geq 3$ in $O(n^{2-\delta})$ time, with $\delta>0$.*

Similar hypotheses have been used in recent works [5, 37, 10, 40]. The main difference is that we require all nodes in $V_1$ to be in cycles; such variants of hardness assumptions that are obtained by changing a quantifier in the definition of the problem are popular, see e.g. [4, 16, 1].

▶ **Theorem 4.** *Under Hypothesis 3, for all $\varepsilon,\delta>0$, no algorithm can $5/3-\varepsilon$ approximate the roundtrip diameter of a sparse directed unweighted graph in $O(n^{2-\delta})$ time.*

We are thus left with a gap between the linear time factor-2 upper bound and the subquadratic factor-5/3 lower bound. A related problem with a similar situation is the problem of computing the eccentricity of all nodes in an undirected graph [4]; there, 5/3 is the right number because one can indeed compute a 5/3-approximation in subquadratic time [19]. Could it be the same here?

Alas, our final result is a reduction from the following classical problem in geometry to roundtrip diameter, establishing a barrier for any better-than-2 approximation in subquadratic time.

▶ **Definition 5** (Approximate $\ell_\infty$ Closest-Pair). *Let $\alpha>1$. The $\alpha$-approximate $\ell_\infty$ Closest-Pair (CP) problem is, given $n$ vectors $v_1,\ldots,v_n$ of some dimension $d$ in $\mathbb{R}^n$, determine if there exists $v_i$ and $v_j$ with $\|v_i-v_j\|_\infty\leq 1$, or if for all $v_i$ and $v_j$, $\|v_i-v_j\|_\infty\geq\alpha$.*

Closest-pair problems are well-studied in various metrics; the main question being whether the naive $n^2$ bound can be broken (when $d$ is assumed to be $n^{o(1)}$). For $\ell_\infty$ specifically, a simple reduction from OV proves a quadratic lower bound for $(2-\varepsilon)$-approximations [31]; but going beyond this factor with current reduction techniques runs into a well-known "triangle-inequality" barrier (see [42, 33]). This leaves a huge gap from the upper bounds that can only achieve $O(\log\log n)$ approximations in subquadratic time [31]. Cell-probe lower bounds for the related nearest-neighbors problem suggest that this log-log bound may be optimal [11]; if indeed constant approximations are impossible in subquadratic time then the following theorem implies a tight lower bound for roundtrip diameter.

▶ **Theorem 6.** *If for some $\alpha\geq 2,\varepsilon>0$ there is a $2-\frac{1}{\alpha}-\varepsilon$ approximation algorithm in time $O(m^{2-\varepsilon})$ for roundtrip diameter in unweighted graphs, then for some $\delta>0$ there is an $\alpha$-approximation for $\ell_\infty$-Closest-Pair with vectors of dimension $d\leq n^{1-\delta}$ in time $\tilde{O}(n^{2-\delta})$.*

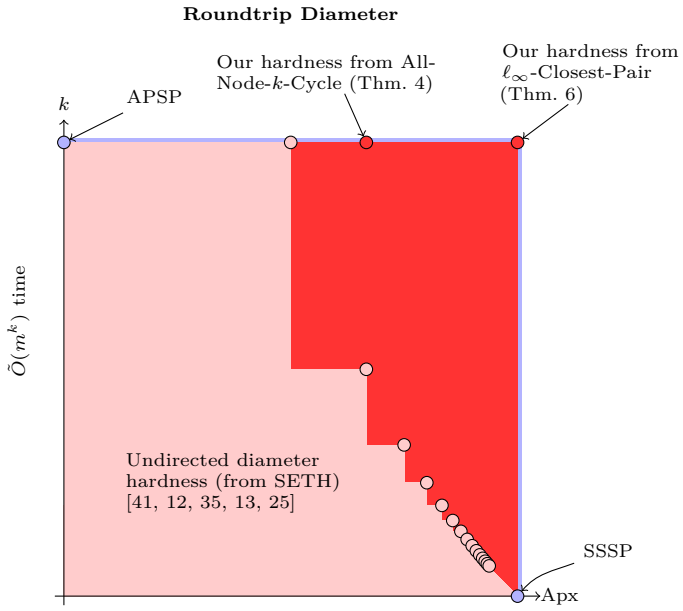



**Figure 3** Roundtrip Diameter algorithms and hardness. All tradeoffs hold for both weighted and unweighted graphs (though citations may differ for weighted vs. unweighted). The previously best hardness results were those inherited from undirected diameter.

In particular, a $2-\varepsilon$ approximation for roundtrip diameter in subquadratic time implies an $\alpha$-approximation for the $\ell_\infty$-Closest-Pair problem in subquadratic time, for some $\alpha = O(1/\varepsilon)$. Thus, any further progress on the roundtrip diameter problem requires a breakthrough on one of the most basic algorithmic questions regarding the $\ell_\infty$ metric (see Figure 3).

## 1.1 Related Work

Besides the diameter and the roundtrip diameter, there is another natural version of the diameter problem in directed graphs called Min-Diameter [4, 27, 24]. The distance between $u, v$ is defined as the $\min(d(u,v), d(v,u))$.[3] This problem seems to be even harder than roundtrip because even a 2-approximation in subquadratic time is not known.

The fine-grained complexity results on diameter (in the sequential setting) have had interesting consequences for computing the diameter in distributed settings (specifically in the CONGEST model). Techniques from both the approximation algorithms and from the hardness reductions have been utilized, see e.g. [39, 2, 9]. It would be interesting to explore the consequences of our techniques on the intriguing gaps in that context [29].

## 1.2 Organization

In this extended abstract, we highlight the key ideas in some of our main results (Theorem 1 and Theorem 6) by proving an "easy version" of each theorem. The full proofs of all the results are in the full version of our paper. First, we establish some preliminaries in Section 2. In Section 3, we prove the special case of Theorem 1 when $t = 0$, giving a 7/4-approximation

[3] Note that the Max-Diameter version where we take the max rather than the min is equal to the one-way version.

of the diameter in directed unweighted graphs in time $O(m^{1.458})$. In Section 4.1 we give an overview of the hardness reductions. In Section 4.2, we prove a weakening of Theorem 6 that only holds for weighted graphs.

## 2 Preliminaries

All logs are base $e$ unless otherwise specified. For reals $a \geq 0$, let $[\pm a]$ denote the real interval $[-a, a]$. For a boolean statement $\varphi$, let $\mathbf{1}[\varphi]$ be 1 if $\varphi$ is true and 0 otherwise.

For a vertex $v$ in a graph, let $\deg(v)$ denote its degree. For $r \geq 0$, let $B_r^{in}(v) = \{u : d(u, v) \leq r\}$ be the in-ball of radius $r$ around $v$, and let $B_r^{out}(v) = \{u : d(v, u) \leq r\}$ be the out-ball of radius $r$ around $v$. For $r \geq 0$, let $B_r^{in+}(v)$ be $B_r^{in}(v)$ and their in-neighbors, and let $B_r^{out+}(v)$ be $B_r^{out}(v)$ and their out-neighbors.

Throughout, let $\omega \leq 2.3728596$ denote the matrix multiplication constant. We use the following lemma which says that we can multiply sparse matrices quickly.

▶ **Lemma 7** (see e.g. Theorem 2.5 of [32]). *We can multiply a $a \times b$ and a $b \times a$ matrix, each with at most $ac$ nonzero entries, in time $O(ac \cdot a^{\frac{\omega-1}{2}})$.*[4]

We repeatedly use the following standard fact.

▶ **Lemma 8.** *Given two sets $B \subset V$ with $B$ of size $k$ and $V$ of size $2m$, a set of $4(m/k)\log m$ uniformly random elements of $V$ contains an element of $B$ with probability at least $1 - \frac{1}{m^2}$.*

**Proof.** The probability that $B$ is not hit is $(1 - \frac{k}{2m})^{4m/k \log m} \leq e^{-2\log m} = \frac{1}{m^2}$. ◀

## 3 7/4-approximation of directed (one-way) diameter

In this section, we prove Theorem 1 in the special case of $t = 0$ and unweighted graphs. That is, we give a 7/4-approximation of the (one-way) diameter of a directed unweighted graph in $O(m^{1.4575})$ time. For the rest of this section, let $\alpha = \frac{\omega+1}{\omega+5} \leq 0.4575$.

Before stating the algorithm and proof, we highlight how our algorithm differs from the undirected algorithm of [17]. At a very high level, all known diameter approximation algorithms compute some pairs of distances, and use the triangle inequality to infer other distances, saving runtime. Approximating diameter in directed graphs is harder than in undirected graphs because distances are not symmetric, so we can only use the triangle inequality "one way." For example, we always have $d(x, y) + d(y, z) \geq d(x, z)$, but not necessarily $d(x, y) + d(z, y) \geq d(x, z)$. The undirected algorithm [17] crucially uses the triangle inequality "both ways," so it was not clear whether their algorithm could be adapted to the directed case. We get around this barrier using matrix multiplication together with the triangle inequality to infer distances quickly. We consider the use of matrix multiplication particularly interesting because, previously, matrix multiplication had only been used for diameter in dense graphs, but we leverage it in sparse graphs.

▶ **Theorem 9.** *Let $\alpha = \frac{\omega+1}{\omega+5}$. There exists a randomized 7/4-approximation algorithm for the diameter of an unweighted directed graph running in $\tilde{O}(m^{1+\alpha})$ time.*

[4] In [32], this runtime of $O(ac \cdot a^{\frac{\omega-1}{2}})$ is stated only for the case $ac > a^{(\omega+1)/2}$. However, the runtime bound for this case works for other cases as well so the lemma is correct for all matrices.

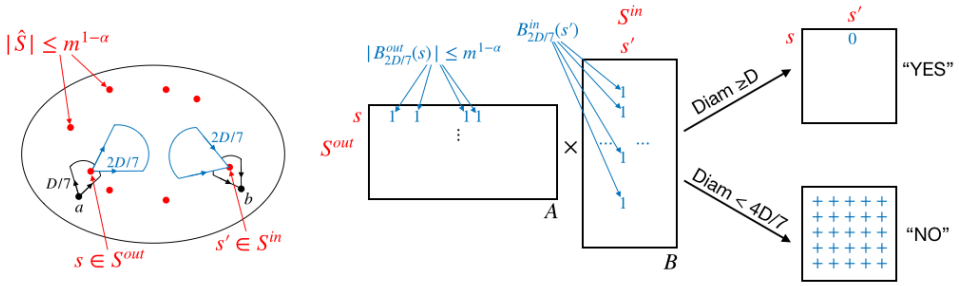



**Figure 4** Steps 5 and 6. If $d(a,b) \ge D$ and Steps 2, 3, and 4 do not accept, with high probability, set $\hat{S}$ hits the $D/7$ out- and in- neighborhoods of $a$ and $b$ at vertices $s$ and $s'$, respectively, that must have distance at least $5D/7$ by the triangle inequality. Thus, checking all pairs of distances in $S^{out} \times S^{in}$, which can be done quickly with sparse matrix multiplication, distinguishes at Step 6 whether the diameter is at least $D$ or less than $4D/7$.

**Proof.** It suffices to show that, for any positive integer $D > 0$, there exists an algorithm $\mathcal{A}_D$ running in time $\tilde{O}(m^{1+\alpha})$ that takes as input any graph and accepts if the diameter is at least $D$, rejects if the diameter is less than $4D/7$, and returns arbitrarily otherwise. Then, we can find the diameter up to a factor of $7/4$ by running binary search with $\mathcal{A}_D$,[5] which at most adds a factor of $O(\log n)$.

We now describe the algorithm $\mathcal{A}_D$. The last two steps, illustrated in Figure 4 contain the key new ideas.

1. First, we apply a standard trick that replaces the input graph on $n$ vertices and $m$ edges with an $2m$-vertex graph of max-degree-3 that preserves the diameter: replace each vertex $v$ with a $\deg(v)$-vertex cycle of weight-0 edges and where the edges to $v$ now connect to distinct vertices of the cycle. From now on, we work with this max-degree-3 graph on $2m$ vertices.
2. Sample $4m^{\alpha} \log m$ uniformly random vertices and compute each vertex's in- and out-eccentricity. If any such vertex has (in- or out-) eccentricity at least $4D/7$ Accept.
3. For every vertex $v$, determine if $|B^{out}_{D/7}(v)| \le m^{\alpha}$. If such a vertex $v$ exists, determine if any vertex in $B^{out+}_{D/7}(v)$ has eccentricity at least $4D/7$, and Accept if so.
4. For every vertex $v$, determine if $|B^{in}_{D/7}(v)| \le m^{\alpha}$. If such a vertex $v$ exists, determine if any vertex in $B^{in+}_{D/7}(v)$ has eccentricity at least $4D/7$, and Accept if so.
5. Sample $4m^{1-\alpha} \log m$ uniformly random vertices $\hat{S}$. Let $S^{out} = \{s \in \hat{S} : |B^{out}_{2D/7}(s)| \le m^{1-\alpha}\}$ and $S^{in} = \{s \in \hat{S} : |B^{in}_{2D/7}(s)| \le m^{1-\alpha}\}$. Compute $B^{out}_{2D/7}(s)$ and $B^{out+}_{2D/7}(s)$ for $s \in S^{out}$, and $B^{in}_{2D/7}(s)$ and $B^{in+}_{2D/7}(s)$ for $s \in S^{in}$.
6. Let $A^{out} \in \mathbb{R}^{S^{out} \times V}$ be the $|S^{out}| \times n$ matrix where $A_{s,v} = \mathbf{1}[v \in B^{out}_{2D/7}(s)]$. Let $A^{in} \in \mathbb{R}^{V \times S^{in}}$ be the $n \times |S^{in}|$ matrix where $A^{in}_{v,s} = \mathbf{1}[v \in B^{in}_{2D/7}(s)]$ if $\lfloor 4D/7 \rfloor = 2\lfloor 2D/7 \rfloor$ and $A^{in}_{v,s} = \mathbf{1}[v \in B^{in+}_{2D/7}(s)]$ otherwise. Compute $A^{out} \cdot A^{in} \in \mathbb{R}^{S^{out} \times S^{in}}$ using sparse matrix multiplication. If the product has any zero entries, Accept, otherwise Reject.

[5] We have to be careful not to lose a small additive factor. Here are the details: Let $D^*$ be the true diameter. Initialize $hi = n, lo = 0$. Repeat until $hi - lo = 1$: let $mid = \lfloor (hi + lo)/2 \rfloor$, run $\mathcal{A}_{mid}$, if accept, set $lo = mid$, else $hi = mid$. One can check that $hi \ge D^* + 1$ and $lo \le 7D^*/4$ always hold. If we return $lo$ after the loop breaks, the output is always in $[D^*, 7D^*/4]$.

**Runtime.** Computing a single eccentricity takes time $O(m)$, so Step 2 takes time $\tilde{O}(m^{1+\alpha})$. For Step 3 checking if $|B^{out}_{D/7}(v)| \leq m^\alpha$ takes $O(m^\alpha)$ time for each $v$ via a partial Breadth-First-Search (BFS). Here we use that the max-degree is 3. If $|B^{out}_{D/7}(v)| \leq m^\alpha$, there are at most $3m^\alpha$ eccentricity computations which takes time $O(m^{1+\alpha})$. Step 4 takes time $O(m^{1+\alpha})$ for the same reason. Similarly, we can complete Step 5 by running partial BFS for each $s \in \hat{S}$ until $m^{1-\alpha}$ vertices are visited. This gives $S^{out}$ and $S^{in}$ and also gives $B^{out}_{2D/7}(s)$ and $B^{out+}_{2D/7}(s)$ for $s \in S^{out}$ and $B^{in}_{2D/7}(s)$ and $B^{in+}_{2D/7}(s)$ for $s \in S^{in}$. For Step 6, the runtime is the time to multiplying sparse matrices. Matrix $A^{out}$ has at most $|\hat{S}| \leq 4m^{1-\alpha}\log m$ rows each with at most $\max_{s\in S^{out}} |B^{out}_{2D/7}(s)| \leq m^{1-\alpha}$ entries, and similarly $A^{in}$ has at most $4m^{1-\alpha}\log m$ columns each with at most $\max_{s\in S^{in}} |B^{in+}_{2D/7}(s)| \leq 3m^{1-\alpha}$ entries. The sparse matrix multiplication takes time $\tilde{O}(m^{(2-2\alpha)} \cdot m^{(1-\alpha)\frac{\omega-1}{2}}) = \tilde{O}(m^{1+\alpha})$ by Lemma 7 with $a = m^{1-\alpha}, b = n, c = m^{1-\alpha}$.

**If the Diameter is less than $4D/7$, we always reject.** Clearly every vertex has eccentricity less than $4D/7$, so we indeed do not accept at Steps 2, 3, and 4. In Step 5, we claim for every $s \in S^{out}, s' \in S^{in}$ there exists $v$ such that $A^{out}_{s,v} = A^{in}_{v,s'} = 1$, so that $(A^{out} \cdot A^{in})_{s,s'} \geq 1$ for all $s \in S^{out}$ and $s' \in S^{in}$ and thus we reject. Fix $s \in S^{out}$ and $s' \in S^{in}$. By the diameter bound, $d(s, s') \leq \lfloor 4D/7 \rfloor$. Let $v$ be the last vertex on the $s$-to-$s'$ shortest path such that $d(s, v) \leq \lfloor 2D/7 \rfloor$, and, if it exists, let $v'$ be the vertex after $v$. Clearly $A^{out}_{s,v} = 1$. We show $A^{in}_{v,s'} = 1$ as well. If $v = s'$, then clearly $v \in B^{in}_{2D/7}(s')$ so $A^{in}_{v,s'} = 1$ as desired. Otherwise $d(s, v) = \lfloor 2D/7 \rfloor$. If $\lfloor 4D/7 \rfloor = 2\lfloor 2D/7 \rfloor$, then $d(v, s') \leq d(s, s') - d(s, v) \leq \lfloor 4D/7 \rfloor - \lfloor 2D/7 \rfloor = \lfloor 2D/7 \rfloor$, so $v \in B^{in}_{2D/7}(s')$ and $A^{in}_{v,s'} = 1$, so again $A^{in}_{v,s'} = 1$. If $\lfloor 4D/7 \rfloor = 2\lfloor 2D/7 \rfloor + 1$, then $d(v', s') \leq d(s, s') - d(s, v') \leq \lfloor 4D/7 \rfloor - (\lfloor 2D/7 \rfloor + 1) = \lfloor 2D/7 \rfloor$, so $v' \in B^{in}_{2D/7}(s')$ and thus $v \in B^{in+}_{2D/7}(s')$ and $A^{in}_{v,s'} = 1$, as desired. This covers all cases, so we've shown we reject.

**If the Diameter is at least $D$, we accept with high probability.** Let $a$ and $b$ be vertices with $d(a, b) \geq D$.

If $|B^{out}_{3D/7}(a)| > m^{1-\alpha}$, Step 2 computes the eccentricity of some $v \in B^{out}_{3D/7}(a)$ with high probability (by Lemma 8), which is at least $d(v, b) \geq d(a, b) - d(a, v) \geq 4D/7$ by the triangle inequality, so we accept. Similarly, we accept with high probability if $|B^{in}_{3D/7}(b)| > m^{1-\alpha}$. Thus we may assume that $|B^{out}_{3D/7}(a)|, |B^{in}_{3D/7}(b)| \leq m^{1-\alpha}$ for the rest of the proof.

If $|B^{out}_{D/7}(v)| \leq m^\alpha$ for any vertex $v$, then either (i) $d(v, b) \geq 4D/7$, in which case $v$ has eccentricity at least $4D/7$ and we accept at Step 3, or (ii) $d(v, b) \leq 4D/7$, in which case there is a vertex $u \in B^{out+}_{D/7}(v)$ on the $v$-to-$b$ path with $d(u, b) \leq 3D/7$ (take the $u \in B^{out+}_{D/7}(v)$ closest to $b$ on the path). Then $d(a, u) \geq 4D/7$ by the triangle inequality and we accept in Step 3 as we perform a BFS from $u$. Thus we may assume $|B^{out}_{D/7}(v)| > m^\alpha$ for all vertices $v$. Similarly, because of Step 4, we may assume $|B^{in}_{D/7}(v)| > m^\alpha$ for all vertices $v$.

In particular, we may assume $|B^{out}_{D/7}(a)| > m^\alpha$ and $|B^{in}_{D/7}(b)| > m^\alpha$. Figure 4 illustrates this last step. Then $\hat{S}$ hits $B^{out}_{D/7}(a)$ with high probability (by Lemma 8), so $B^{out}_{D/7}(a)$ has some $s \in \hat{S}$ with high probability, and similarly $B^{in}_{D/7}(b)$ has some $s' \in \hat{S}$ with high probability. The triangle inequality implies that $B^{out}_{2D/7}(s) \subset B^{out}_{3D/7}(a)$, so $|B^{out}_{2D/7}(s)| \leq |B^{out}_{3D/7}(a)| \leq m^{1-\alpha}$ and thus $s \in S^{out}$. Similarly $s' \in S^{in}$. By the triangle inequality, we have $d(s, s') \geq d(a, b) - d(a, s) - d(s', b) \geq D - D/7 - D/7 = 5D/7$. Then we must have $(A \cdot B)_{s,s'} = 0$, as otherwise there is a $v$ such that $d(s, v) \leq \lfloor 2D/7 \rfloor$ and $d(v, s') \leq 4D/7 - \lfloor 2D/7 \rfloor$, contradicting $d(s, s') \geq 5D/7$. Hence, we accept at step 5, as desired. ◀

# 4 Hardness Reductions for Roundtrip

## 4.1 Overview

In this paper we prove hardness results for roundtrip diameter that go beyond the 2 vs. 3 barrier. Before presenting the proofs, let us begin with an abstract discussion on why this barrier arises and (at a high level) how we overcome it.

All previous hardness results for diameter are by reductions from OV (or its generalization to multiple sets). In OV, one is given two sets of vectors of size $n$ and dimension $d = \text{poly} \log n$, $A$ and $B$, and one needs to determine whether there are $a \in A, b \in B$ that are orthogonal. SETH implies that OV requires $n^{2-o(1)}$ time [46]. In a reduction from OV to a problem like diameter, one typically has nodes representing the vectors in $A$ and $B$, as well as nodes $C$ representing the coordinates, and if there is an orthogonal vector pair $a, b$, then the corresponding nodes in the diameter graph are far (distance $\geq 3$), and otherwise *all* pairs of nodes are close (distance $\leq 2$). Going beyond the 2 vs. 3 gap is difficult because each node $a \in A$ must have distance $\leq 2$ to each coordinate node in $C$, regardless of the existence of an orthogonal pair, and then it is automatically at distance $2 + 1$ from any node $b \in B$ because each $b$ has at least one neighbor in $C$. So even if $a, b$ are orthogonal, the distance will not be more than 3.

The key trick for proving a higher lower bound (say 3 vs. 5) for roundtrip is to have two sets of coordinate nodes, a $C^{fwd}$ set that can be used to go *forward* from $A$ to $B$, and a $C^{bwd}$ set that can be used to go back. The default roundtrip paths from $A/B$ to each of these two sets will have different forms, and this asymmetry will allow us to overcome the above issue. This is inspired by the difficulty that one faces when trying to make the subquadratic 3/2-approximation algorithms for undirected and directed diameter work for roundtrip.

Unfortunately, there is another (related) issue when reducing from OV. First notice that all nodes within $A$ and within $B$ must always have small distance (or else the diameter would be large). This can be accomplished simply by adding direct edges of weight 1.5 between all pairs (within $A$ and within $B$); but this creates a dense graph and makes the quadratic lower bound uninteresting. Instead, such reductions typically add auxiliary nodes to simulate the $n^2$ edges more cheaply, e.g. a star node $o$ that is connected to all of $A$. But then the node $o$ must have small distance to $B$, decreasing all distances between $A$ and $B$.

Overcoming this issue by a similar trick seems impossible. Instead, our two hardness results bypass it in different ways.

The reduction from $\ell_\infty$-Closest-Pair starts from a problem that is defined over *one set of vectors* $A$ (not two) which means that the coordinates are "in charge" of connecting all pairs within $A$. We remark that while OV can also be defined over one set (monochromatic) instead of two (bichromatic) and that it remains SETH hard; that would prevent us from applying the above trick of having a forward and a backward sets of coordinate nodes. Our reduction in Section 4.2 is able to utilize the structure of the metric in order to make both ideas work simultaneously.

The reduction from All-Node $k$-Cycle relies on a different idea: it uses a construction where only a small set of $n$ pairs $a_i \in A, b_i \in B$ are "interesting" in the sense that we do not care about the distances for other pairs (in order to solve the starting problem). Then the goal becomes to connect all pairs within $A$ and within $B$ by short paths, without decreasing the distance for the $(a_i, b_i)$ pairs. A trick similar to the *bit-gadget* [3, 2] does the job. For the complete reduction see the full version of the paper.

## 4.2 Weighted Roundtrip $2-\varepsilon$ hardness from $\ell_\infty$-CP

In this section, we highlight the key ideas in Theorem 6 by proving a weaker version, showing the lower bound for weighted graphs. See the full version of the paper for the extension to unweighted graphs.

The main technical lemma is showing that to $\alpha$-approximate $\ell_\infty$-Closest-Pair, it suffices to do so on instances where all vector coordinates are in $[\pm(0.5+\varepsilon)\alpha]$. Towards this goal, we make the following definition.

▶ **Definition 10.** *The $\alpha$-approximate $\beta$-*bounded *$\ell_\infty$-Closest-Pair problem is, given $n$ vectors $v_1,\ldots,v_n$ of dimension $d$ in $[-\beta,\beta]^d$ determine if there exists $v_i$ and $v_j$ with $\|v_i-v_j\|_\infty\le 1$, or if for all $v_i$ and $v_j$, $\|v_i-v_j\|_\infty\ge\alpha$.*

We now prove the main technical lemma.

▶ **Lemma 11.** *Let $\varepsilon\in(0,1/2)$ and $\alpha>1$. If one can solve $\alpha$-approximate $(0.5+\varepsilon)\alpha$-bounded $\ell_\infty$-CP on dimension $O(d\varepsilon^{-1}\log n)$ in time $T$, then one can solve $\alpha$-approximate $\ell_\infty$-CP on dimension $d$ in time $T+O_\varepsilon(dn\log n)$, where in $O_\varepsilon(\cdot)$ we neglect dependencies on $\varepsilon$.*

**Proof.** Start with an $\ell_\infty$ instance $\Phi=(v_1,\ldots,v_n)$. We show how to construct a bounded $\ell_\infty$ instance $\Phi'$ such that $\Phi$ has two vectors with $\ell_\infty$ distance $\le 1$ if and only if $\Phi'$ has two vectors with $\ell_\infty$ distance $\le 1$.

First we show we may assume that $v_1,\ldots,v_n$ are on domain $[0,\alpha n]$. Suppose that $x\in[d]$. Reindex $v_1,\ldots,v_n$ in increasing order of $v_i[x]$ (by sorting). Let $v_1',\ldots,v_n'$ be vectors identical to $v_1,\ldots,v_n$ except in coordinate $x$, where instead

$$v_i'[x]=\sum_{j=0}^{i-1}\min(\alpha,v_{j+1}[x]-v_j[x])$$

for $i=1,\ldots,n$, where the empty sum is 0. We have that $v_i'[x]\le\alpha n$ for all $i$, and furthermore $|v_i'[x]-v_j'[x]|\ge\alpha$ if and only if $|v_i[x]-v_j[x]|\ge\alpha$ and also $|v_i'[x]-v_j'[x]|\le 1$ if and only if $|v_i[x]-v_j[x]|\le 1$. Hence, the instance given by $v_1',\ldots,v_n'$ is a YES instance if and only if the instance $\Phi$ is a YES instance, and is a NO instance if and only if the instance $\Phi$ is a NO instance. Repeating this with all other coordinates $x$ gives an instance $\Phi'$ such that $\Phi'$ is a YES instance if and only if $\Phi$ is a YES instance, and $\Phi'$ is a NO instance if and only if $\Phi'$ is a NO instance, and furthermore $\Phi'$ has vectors on $[0,\alpha n]$.

Now we show how to construct an $\ell_\infty$-CP instance in dimension $O_\varepsilon(d\log n)$ vectors with coordinates in $[\pm(0.5+\varepsilon)\alpha]$.

▶ **Lemma 12.** *Let $\varepsilon\in(0,0.5)$ and $\alpha>1$. For any real number $M$, there exists two maps $g:[0,M]\to[-(0.5+\varepsilon)\alpha,(0.5+\varepsilon)\alpha]^{2\lceil\varepsilon^{-1}\rceil+1}$ and $h:[0,M]\to[0,M/2]$ such that for all $a,b\in[0,M]$, we have $\min(|a-b|,\alpha)=\min(\|(g(a),h(a))-(g(b),h(b))\|_\infty,\alpha)$. (here, $(g(\cdot),h(\cdot))$ is a length $2\lceil\varepsilon^{-1}\rceil+2$ vector.) Furthermore, $g$ and $h$ can be computed in $O_\varepsilon(1)$ time.*

**Proof.** It suffices to consider when $\varepsilon^{-1}$ is an integer. Let $f_z:\mathbb{R}\to[-(0.5+\varepsilon)\alpha,(0.5+\varepsilon)\alpha]$ be the piecewise function

$$f_z(x)=\begin{cases}-(0.5+\varepsilon)\alpha & \text{if } x\le z-(0.5+\varepsilon)\alpha\\ (0.5+\varepsilon)\alpha & \text{if } x\ge z+(0.5+\varepsilon)\alpha\\ x-z & \text{otherwise.}\end{cases}$$

For $a \in [M]$, define $g(a) \in \mathbb{R}^{2\varepsilon^{-1}+1}$ and $h(a) \in \mathbb{R}$ as follows, where we index coordinates by $-\varepsilon^{-1}, \dots, -1, 0, 1, \varepsilon^{-1}$ for convenience

$$g(a)_i = f_{M/2+0.5i\varepsilon\alpha}(a) \text{ for } -\varepsilon^{-1} \le i \le \varepsilon^{-1}$$
$$h(a) = |a - M/2|.$$

Clearly $g$ and $h$ have the correct codomain, and they can be computed in $O_\varepsilon(1)$ time. Additionally, note that $f_z(x)$ and $|x - M/2|$ are 1-Lipschitz functions of $x$ for all $z$, so $g$ is a Lipschitz function and thus $\|g(a) - g(b)\|_\infty \le |a - b|$.

Now, it suffices to show that $\min(\|(g(a), h(a)) - (g(b), h(b))\|_\infty, \alpha) \ge \min(|a - b|, \alpha)$. If $a$ and $b$ are on the same side of $M/2$, then $\|h(a) - h(b)\|_\infty \ge ||a - M/2| - |b - M/2|| = |a - b|$, as desired. Now suppose $a$ and $b$ are on opposite sides of $M/2$, and without loss of generality $a < M/2 < b$. Let $0 \le i \le \varepsilon^{-1}$ be the largest integer such that $a \le M/2 - i\varepsilon\alpha$ ($i = 0$ works so $i$ always exists). If $i = \varepsilon^{-1}$, then $a < M/2 - \alpha$ and

$$\|g(a) - g(b)\|_\infty \ge f_{M/2-0.5\alpha}(b) - f_{M/2-0.5\alpha}(a) \ge 0.5\alpha - (-0.5\alpha) = \alpha \ge \min(|a - b|, \alpha),$$

as desired. Now assume $i < \varepsilon^{-1}$. Let $z = M/2 + (0.5 - i\varepsilon)\alpha$. By maximality of $i$, we have $a - z \in [-(0.5+\varepsilon)\alpha, -0.5\alpha]$. We have $g(\cdot)_{\varepsilon^{-1}-2i} = f_z(\cdot)$ by definition of $g$. By the definition of $f_z(\cdot)$, since $a \in [z-(0.5+\varepsilon)\alpha, z-0.5\alpha]$ and $b \ge a$, we have $\min(f_z(b)-f_z(a), \alpha) = \min(b-a, \alpha)$. Thus,

$$\begin{aligned}\min(\|g(a) - g(b)\|_\infty, \alpha) &\ge \min\left(g(b)_{\varepsilon^{-1}-2i} - g(a)_{\varepsilon^{-1}-2i}, \alpha\right) \\ &= \min(f_z(b) - f_z(a), \alpha) = \min(b - a, \alpha),\end{aligned}$$

as desired. In either case, we have $\min(\|g(a) - g(b)\|_\infty, \alpha) \ge \min(|a - b|, \alpha)$, so we conclude that $\min(\|g(a) - g(b)\|_\infty, \alpha) = \min(|a - b|, \alpha)$. ◀

Iterating Lemma 12 gives the following.

▶ **Lemma 13.** *Let $\varepsilon \in (0, 1/2)$. There exists a map $g : [0, \alpha n] \to [\pm(0.5+\varepsilon)\alpha]^{4\lceil\varepsilon^{-1}\rceil \log n}$ such that for all $a, b \in [0, \alpha n]$, we have $\min(|a - b|, \alpha) = \min(\|g(a) - g(b)\|_\infty, \alpha)$. Furthermore, $g$ can be computed in $O_\varepsilon(\log n)$ time.*

**Proof.** For $\ell = 1, \dots,$ let $M_\ell = \alpha n/2^{\ell-1}$, and let $g^*_\ell : [M_\ell] \to [\pm(0.5 + \varepsilon)\alpha]^{2\lceil\varepsilon^{-1}\rceil+1}$ and $h^*_\ell : [M_\ell] \to [M_{\ell+1}]$ be the functions given by Lemma 12. For $\ell = 0, 1, \dots,$ let $g_\ell : [0, \alpha n] \to [-(0.5 + \varepsilon)\alpha, (0.5 + \varepsilon)\alpha]^{\ell(2\lceil\varepsilon^{-1}\rceil+1)}$ and $h_\ell : [0, \alpha n] \to [0, \alpha n/2^\ell]$ be such that $g_0(x) = ()$ is an empty vector, $h_0(x) = x$ is the identity, and for $\ell \ge 1$, $g_\ell(x) = (g_{\ell-1}(x), g^*_\ell(h_{\ell-1}(x)))$ and $h_\ell(x) = h^*_\ell(h_{\ell-1}(x))$. By Lemma 12, we have that

$$\begin{aligned}&\min\left(\|(g_{\ell-1}(a), h_{\ell-1}(a)) - (g_{\ell-1}(b), h_{\ell-1}(b))\|_\infty, \alpha\right) \\ &= \min\left(\|\big(g_{\ell-1}(a), g^*_\ell(h_{\ell-1}(a)), h^*_\ell(h_{\ell-1}(a))\big) - \big(g_\ell(b), g^*_\ell(h_{\ell-1}(b)), h^*_\ell(h_{\ell-1}(b))\big)\|_\infty, \alpha\right) \\ &= \min\left(\|\big(g_\ell(a), h_\ell(a)\big) - \big(g_\ell(b), h_\ell(b)\big)\|_\infty, \alpha\right)\end{aligned}$$

for all $\ell$. For $\ell = \lceil \log n \rceil$, the vector $g(a) \stackrel{\text{def}}{=} (g_\ell(a), h_\ell(a) - 0.5\alpha)$ has every coordinate in $[\pm(0.5 + \varepsilon)\alpha]$, and by (4.2), we have

$$\begin{aligned}\min(|a - b|, \alpha) &= \min(|g_0(a) - g_0(b)|, \alpha) \\ &= \min(|g_\ell(a) - g_\ell(b)|, \alpha) = \min(|g(a) - g(b)|_\infty, \alpha),\end{aligned}$$

as desired. The length of this vector is at most $\lceil \log n \rceil (2\lceil\varepsilon^{-1}\rceil + 1) + 1$, which we bound by $4\lceil\varepsilon^{-1}\rceil \log n$ for simplicity (and pad the corresponding vectors with zeros). ◀

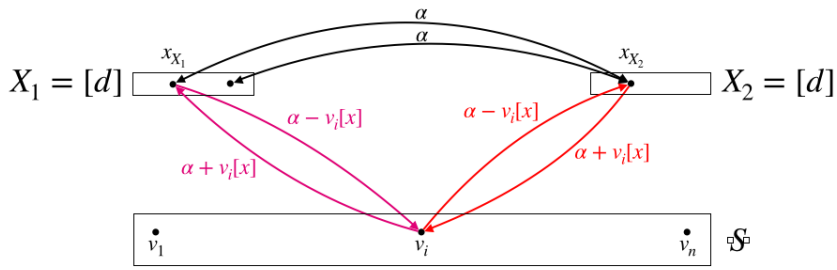



**Figure 5** The roundtrip diameter instance $G$ for $\ell_\infty$-CP hardness.

To finish, let $g : [0, \alpha n] \to [\pm(0.5+\varepsilon)\alpha]$ be given by Lemma 13, and let the original $\ell_\infty$ instance be $v_1, \ldots, v_n$. Let the new $(0.5+\varepsilon)\alpha$-bounded $\ell_\infty$ instance be $w_i = (g(v_i[x]))_{x\in[d]}$ of length $4d\lceil\varepsilon^{-1}\rceil \log n$. ◀

We now prove our goal for this section, Theorem 6 for weighted graphs.

▶ **Theorem 14.** *If for some $\alpha \geq 2, \varepsilon > 0$ there is a $2 - \frac{1}{\alpha} - \varepsilon$ approximation algorithm in time $O(m^{2-\varepsilon})$ for roundtrip diameter in* weighted *graphs, then for some $\delta > 0$ there is an $\alpha$-approximation for $\ell_\infty$-Closest-Pair with vectors of dimension $d \leq n^{1-\delta}$ in time $\tilde{O}(n^{2-\delta})$.*

**Proof.** By Lemma 11 it suffices to prove that there exists an $O(n^{2-\delta})$ time algorithm for $\alpha$-approximate $(0.5+\varepsilon)\alpha$-bounded $\ell_\infty$-CP for $\varepsilon = (4\alpha)^{-1}$.

Let $\Phi$ be the bounded-domain $\ell_\infty$-CP instance with vectors $v_1, \ldots, v_n \in [\pm(0.5+\varepsilon)\alpha]^n$. Then construct a graph $G$ (see Figure 5) with vertex set $S \cup X_1 \cup X_2$ where $X_1 = X_2 = [d]$ and $S = [n]$. We identify vertices with the notations $i_S, x_{X_1}$, and $x_{X_2}$, for $i \in [n]$ and $x \in [d]$. Draw directed edges

1. from $i_S$ to $x_{X_1}$, of weight $\alpha + v_i[x]$,
2. from $x_{X_1}$ to $i_S$, of weight $\alpha - v_i[x]$,
3. from $i_S$ to $x_{X_2}$, of weight $\alpha - v_i[x]$,
4. from $x_{X_2}$ to $i_S$, of weight $\alpha + v_i[x]$, and
5. between any two vertices in $X_1 \cup X_2$, of weight $\alpha$.

Note that all edge weights are nonnegative, and any two vertices in $X_1 \cup X_2$ are roundtrip distance $2\alpha$, and any $s \in S$ and $x \in X_1 \cup X_2$ are distance $2\alpha$. Suppose $\Phi$ has no solution, so that every pair has $\ell_\infty$ distance $\alpha$. Then for vertices $i_S, j_S$, there exists a coordinate $x$ such that $v_i[x] - v_j[x]$ is either $\geq \alpha$ or $\leq -\alpha$. Without loss of generality, we are in the case $v_i[x] - v_j[x] \geq \alpha$. Then the path $i_S \to x_{X_2} \to j_S \to x_{X_1} \to i_S$ is a roundtrip path of length

$$(\alpha - v_i[x]) + (\alpha + v_j[x]) + (\alpha + v_j[x]) + (\alpha - v_i[x]) = 4\alpha - 2(v_i[x] - v_j[x]) \leq 2\alpha.$$

So when $\Phi$ has no solution, the roundrip diameter is at most $2\alpha$.

On the other hand, suppose $\Phi$ has a solution $i, j$ such that for all $x$, $|v_i[x] - v_j[x]| \leq 1$. Then, as every edge has weight at least $(0.5-\varepsilon)\alpha$,

$$\begin{aligned} d(i_S, j_S) &\geq \min\left(\min_{x\in[d]} \left(d(i_S, x_{X_1}) + d(x_{X_1}, j_S), d(i_S, x_{X_2}) + d(x_{X_2}, j_S)\right), 4(0.5-\varepsilon)\alpha\right) \\ &\geq \min\left(\min_{x\in[d]} (\alpha + v_i[x] + \alpha - v_j[x], \alpha + v_j[x] + \alpha - v_i[x]), 2\alpha - 4\varepsilon\alpha\right) \\ &\geq \min(2\alpha - 1, 2\alpha - 4\alpha\varepsilon) \;=\; 2\alpha - 1. \end{aligned}$$

Similarly, we have

$$d(j_S, i_S) \geq 2\alpha - 1,$$

so we have

$$d_{RT}(j_S, i_S) \geq 4\alpha - 2.$$

so in this case the RT-diameter is at least $4\alpha - 2$. A $2 - \alpha^{-1} - \varepsilon$ approximation for RT diameter can distinguish between RT diameter $4\alpha - 2$ and RT-diameter $2\alpha$. Thus, a $2 - \alpha - \varepsilon$ approximation for RT diameter solves $\alpha$-approximate $\ell_\infty$-CP. ◀

## References


1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Scheduling lower bounds via AND subset sum. *J. Comput. Syst. Sci.*, 127:29–40, 2022. doi:10.1016/j.jcss.2022.01.005.

2 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *ACM Trans. Algorithms*, 17(4):30:1–30:40, 2021. doi:10.1145/3469834.

3 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

4 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

5 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 239–252, 2018.

6 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, pages 522–539. SIAM, 2021.

7 N. Alon, Z. Galil, and O. Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.

8 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

9 Bertie Ancona, Keren Censor-Hillel, Mina Dalirrooyfard, Yuval Efron, and Virginia Vassilevska Williams. Distributed distance approximation. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, volume 184 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.OPODIS.2020.30.

10 Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

11 Alexandr Andoni, Dorian Croitoru, and Mihai Patrascu. Hardness of nearest neighbor under l-infinity. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 424–433, 2008. doi:10.1109/FOCS.2008.89.

12 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 267–280. ACM, 2018.

13 Édouard Bonnet. 4 vs 7 sparse undirected unweighted diameter is seth-hard at time $n^{4/3}$. In *Proc. ICALP*, pages 34:1–34:15, 2021.

14 Édouard Bonnet. Inapproximability of diameter in super-linear time: Beyond the 5/3 ratio. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 17:1–17:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

15 Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter A Kosters, Andrea Marino, and Frank W Takes. Fast diameter and radius bfs-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games. *Theoretical Computer Science*, 586:59–80, 2015.

16 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. *J. Comput. Geom.*, 11(2):94–130, 2020. doi:10.20382/jocg.v11i2a5.

17 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376, 2016.

18 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified $k$-cnf. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation – 5th International Symposium, IPEC 2010, Chennai, India, December 13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2010.

19 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052, 2014.

20 Shiri Chechik and Gur Lifshitz. Optimal girth approximation for dense directed graphs. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, pages 290–300. SIAM, 2021.

21 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Constant girth approximation for directed graphs in subquadratic time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1010–1023. ACM, 2020.

22 Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing for digraphs. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 885–886. ACM/SIAM, 1999.

23 Pierluigi Crescenzi, Roberto Grossi, Leonardo Lanzi, and Andrea Marino. On computing the diameter of real-world directed (weighted) graphs. In Ralf Klasing, editor, *Experimental Algorithms: 11th International Symposium, SEA 2012, Bordeaux, France, June 7-9, 2012. Proceedings*, pages 99–110, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

24 Mina Dalirrooyfard and Jenny Kaufmann. Approximation algorithms for min-distance problems in dags. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

25 Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. In *Proc. FOCS*, FOCS'2021, pages 1021–1032, 2021.

26 Mina Dalirrooyfard and Virginia Vassilevska Williams. Conditionally optimal approximation algorithms for the girth of a directed graph. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

27 Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, Nicole Wein, Yinzhan Xu, and Yuancheng Yu. Approximation algorithms for min-distance problems. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

28 Mina Dalirrooyfard and Nicole Wein. Tight conditional lower bounds for approximating diameter in directed graphs. In *Proc. STOC*, STOC'2021, pages 1697–1710, 2021.

29 Ofer Grossman, Seri Khoury, and Ami Paz. Improved hardness of approximation of diameter in the CONGEST model. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.19.

30 R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

31 Piotr Indyk. On approximate nearest neighbors under l-infinity norm. *Journal of Computer and System Sciences*, 63(4):627–638, 2001.

32 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 52–60, 2006.

33 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. *Comb.*, 40(4):539–573, 2020. doi:10.1007/s00493-019-4113-1.

34 Ray Li. Improved seth-hardness of unweighted diameter. *CoRR*, abs/2008.05106v1, 2020. arXiv:2008.05106.

35 Ray Li. Settling seth vs. approximate sparse directed unweighted diameter (up to (nu)nseth). In *Proc. STOC*, STOC'2021, pages 1684–1696, 2021.

36 T. C. Lin, M. J. Wu, W. J. Chen, and B. Y. Wu. Computing the diameters of huge social networks. In *2016 International Computer Symposium (ICS)*, pages 6–11, 2016.

37 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252. SIAM, 2018.

38 Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1374–1392. SIAM, 2018.

39 David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II*, volume 7392 of *Lecture Notes in Computer Science*, pages 660–672. Springer, 2012. doi:10.1007/978-3-642-31585-5_58.

40 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 153–166, 2020.

41 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 515–524, New York, NY, USA, 2013. ACM. doi:10.1145/2488608.2488673.

42 Aviad Rubinstein. Hardness of approximate nearest neighbor search. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1260–1268. ACM, 2018. doi:10.1145/3188745.3188916.

43 Aviad Rubinstein and Virginia Vassilevska Williams. Seth vs approximation. *ACM SIGACT News*, 50(4):57–76, 2019.

44 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

45 Frank W. Takes and Walter A. Kosters. Determining the diameter of small world networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 1191–1196, 2011.

46 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

47 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002. Announced at FOCS'98. doi:10.1145/567112.567114.

# Can You Solve Closest String Faster Than Exhaustive Search?

**Amir Abboud**
Weizmann Institute of Science, Rehovot, Israel

**Nick Fischer**
Weizmann Institute of Science, Rehovot, Israel

**Elazar Goldenberg**
Academic College of Tel Aviv-Yaffo, Israel

**Karthik C. S.**
Rutgers University, New Brunswick, NJ, USA

**Ron Safier**
Weizmann Institute of Science, Rehovot, Israel

## Abstract

We study the fundamental problem of finding the best string to represent a given set, in the form of the *Closest String* problem: Given a set $X \subseteq \Sigma^d$ of $n$ strings, find the string $x^*$ minimizing the radius of the smallest Hamming ball around $x^*$ that encloses all the strings in $X$. In this paper, we investigate whether the Closest String problem admits algorithms that are faster than the trivial exhaustive search algorithm. We obtain the following results for the two natural versions of the problem:

- In the *continuous* Closest String problem, the goal is to find the solution string $x^*$ anywhere in $\Sigma^d$. For binary strings, the exhaustive search algorithm runs in time $O(2^d \operatorname{poly}(nd))$ and we prove that it cannot be improved to time $O(2^{(1-\epsilon)d} \operatorname{poly}(nd))$, for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis fails.
- In the *discrete* Closest String problem, $x^*$ is required to be in the input set $X$. While this problem is clearly in polynomial time, its fine-grained complexity has been pinpointed to be quadratic time $n^{2\pm o(1)}$ whenever the dimension is $\omega(\log n) < d < n^{o(1)}$. We complement this known hardness result with new algorithms, proving essentially that whenever $d$ falls out of this hard range, the discrete Closest String problem can be solved faster than exhaustive search. In the small-$d$ regime, our algorithm is based on a novel application of the inclusion-exclusion principle.

Interestingly, all of our results apply (and some are even stronger) to the natural dual of the Closest String problem, called the *Remotest String* problem, where the task is to find a string maximizing the Hamming distance to all the strings in $X$.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness

**Keywords and phrases** Closest string, fine-grained complexity, SETH, inclusion-exclusion

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.3

**Related Version** *Full Version*: `https://arxiv.org/abs/2305.16878`

**Funding** *Amir Abboud*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No 101078482). Additionally, Amir Abboud is supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.
*Karthik C. S.*: This work was supported by a grant from the Simons Foundation, Grant Number 825876, Awardee Thu D. Nguyen and by the National Science Foundation under Grant CCF-2313372.





# 1 Introduction

The challenge of characterizing a set of strings by a single *representative* string is a fundamental problem all across computer science, arising in essentially all contexts where strings are involved. The basic task is to find a string $x^*$ which minimizes the maximum number of mismatches to all strings in a given set $X$. Equivalently, the goal is to find the center $x^*$ of a smallest ball enclosing all strings in $X$ in the Hamming (or $\ell_0$) metric. This problem has been studied under various names, including *Closest String*, *1-Center in the Hamming metric* and *Chebyshev Radius*, and constitutes the perhaps most elementary clustering task for strings.

In the literature, the Closest String problem has received a lot of attention [11, 12, 23, 14, 22, 26, 9, 25, 20, 27, 1], and it is not surprising that besides the strong theoretical interest, it finds wide-reaching applications in various domains including machine learning, bioinformatics, coding theory and cryptography. One such application in machine learning is for clustering *categorical* data. Typical clustering objectives involve finding good center points to characterize a set of feature vectors. For numerical data (such as a number of publications) this task translates to a center (or median) problem over, say, the $\ell_1$ metric which can be solved using geometry tools. For categorical data, on the other hand, the points have non-numerical features (such as blood type or nationality) and the task becomes finding a good center string over the Hamming metric.

Another important application, in the context of computational biology, is the computer-aided design of PCR primers [25, 24, 10, 29, 13, 32]. On a high level, in the PCR method the goal is to find and amplify (i.e., copy millions of times) a certain fragment of some sample DNA. To this end, short DNA fragments (typically 18 to 25 nucleotides) called *primers* are used to identify the start and end of the region to be copied. These fragments should match as closely as possible the target regions in the sample DNA. Designing such primers is a computational task that reduces exactly to finding a closest string in a given set of genomes.

The Closest String problem comes in two different flavors: In the *continuous* Closest String problem the goal is to select an *arbitrary* center string $x^* \in \Sigma^d$ (here, $\Sigma$ is the underlying alphabet) that minimizes the maximum Hamming distance to the $n$ strings in $X$. This leads to a baseline algorithm running in exponential time $O(|\Sigma|^d \operatorname{poly}(nd))$. In the *discrete* Closest String problem, in contrast, the task is to select the best center $x^*$ in the given set of strings $X$; this problem therefore admits a baseline algorithm in time $O(n^2 d)$. Despite the remarkable attention that both variants have received so far, the most basic questions about the continuous and discrete Closest String problems have not been fully resolved yet:

*Can the $O(|\Sigma|^d \operatorname{poly}(nd))$-time algorithm for continuous Closest String be improved?*
*Can the $O(n^2 d)$-time algorithm for discrete Closest String be improved?*

In this paper, we make considerable progress towards resolving both driving questions, by respectively providing tight conditional lower bounds and new algorithms. In the upcoming Sections 1.1 and 1.2 we will address these questions in depth and state our results.

Interestingly, in both cases our results also extend, at times even in a stronger sense, to a natural dual of the Closest String problem called the *Remotest String* problem. Here, the task is to find a string $x^*$ that maximizes the minimum Hamming distance from $x^*$ to a given set of strings $X$. This problem has also been studied in computational biology [22, 21] and more prominently in the context of coding theory: The remotest string distance is a fundamental parameter of any code which is also called the *covering radius* [8], and under this

name the Remotest String problem has been studied in previous works [28, 15, 6, 17] mostly for specific sets $X$ such as linear codes or lattices. See Alon, Panigrahy and Yekhanin [6] for further connections to matrix rigidity.

## 1.1 Continuous Closest/Remotest String

Let us start with the more classical *continuous* Closest String problem. It is well-known that the problem is NP-complete [11, 22], and up to date the best algorithm remains the naive one: Exhaustively search through all possible strings in time $O(|\Sigma|^d \operatorname{poly}(nd))$. This has motivated the study of approximation algorithms leading to various approximation schemes [12, 23, 25, 27], and also the study through the lens of parameterized algorithms [14]. In this work, we insist on exact algorithms and raise again the question: *Can you solve the continuous Closest String problem faster than exhaustive search?*

For starters, focus on the Closest String problem for *binary* alphabets (i.e., for $|\Sigma| = 2$) which is of particular importance in the context of coding theory [20]. From the known NP-hardness reduction which is based on the 3-SAT problem [11], it is not hard to derive a $2^{d/2}$ lower bound under the Strong Exponential Time Hypothesis (SETH) [18, 19]. This bound clearly does not match the upper bound and possibly leaves hope for a meet-in-the-middle-type algorithm. In our first contribution we shatter all such hopes by strengthening the lower bound, with considerably more effort, to match the time complexity of exhaustive search:

▶ **Theorem 1** (Continuous Closest String is SETH-Hard). *The continuous Closest String problem cannot be solved in time $O(2^{(1-\epsilon)d} \operatorname{poly}(n))$, for any $\epsilon > 0$, unless SETH fails.*

Interestingly, we obtain this lower bound as a corollary of the analogous lower bound for the continuous Remotest String problem (see the following Theorem 2). This is because both problems are equivalent over the binary alphabet. However, even for larger sized alphabet sets $\Sigma$, we obtain a matching lower bound against the Remotest String problem:

▶ **Theorem 2** (Continuous Remotest String is SETH-Hard). *The continuous Remotest String problem cannot be solved in time $O(|\Sigma|^{(1-\epsilon)d} \operatorname{poly}(n))$, for any $\epsilon > 0$ and $|\Sigma| = o(d)$, unless SETH fails.*

Theorem 2 gives a *tight* lower bound for the continuous Remotest String problem in all regimes where we can expect lower bounds, and we therefore close the exact study of the continuous Remotest String problem. Indeed, in the regime where the alphabet size $|\Sigma|$ exceeds the dimension $d$, the Closest and Remotest String problems *can* be solved faster in time $O(d^d \operatorname{poly}(n, d))$ (and even faster parameterized in terms of the target distance [14]).

The intuition behind Theorem 2 is simple: We encode a $k$-SAT instance as a Remotest String problem by viewing strings as *assignments* and by searching for a string which is remote from all *falsifying* assignments. The previously known encoding [11] was inefficient (encoding a single variable $X_i$ accounted for *two* letters in the Remotest String instance: one for encoding the truth value and another one as a "don't care" value for clauses not containing $X_i$), and our contribution is that we make the encoding lossless. While superficially simple, this baseline idea requires a lot of technical effort.

## 1.2 Discrete Closest/Remotest String

Recall that in the *discrete* Closest String problem (in contrast to the continuous one) the solution string $x^*$ must be part of the input set $X$. For applications in the context of data compression and summarization, the discrete problem is often the better choice: Selecting the representative string from a set of, say, grammatically or semantically meaningful strings is typically more informative than selecting an arbitrary representative string.

The problem can be naively solved in time $O(n^2 d)$ by exhaustive search: Compute the Hamming distance between all $\binom{n}{2}$ pairs of strings in $X$ in time $O(d)$ each. In terms of exact algorithms, this running time is the fastest known. Toward our second driving question, we investigate whether this algorithm can be improved, at least for some settings of $n$ and $d$. In previous work, Abboud, Bateni, Cohen-Addad, Karthik, and Seddighin [1] have established a conditional lower bound under the Hitting Set Conjecture [3], stating that the problem requires quadratic time in $n$ whenever $d = \omega(\log n)$:

▶ **Theorem 3** (Discrete Closest String for Super-Logarithmic Dimensions [1]). *The discrete Closest String problem in dimension $d = \omega(\log n)$ cannot be solved in time $O(n^{2-\epsilon})$, for any $\epsilon > 0$, unless the Hitting Set Conjecture fails.*

This hardness result implies that there is likely no polynomially faster algorithm for Closest String whenever the dimension $d$ falls in the range $\omega(\log n) < d < n^{o(1)}$. But this leaves open the important question of whether the exhaustive search algorithm can be improved outside this region, if $d$ is very small (say, $o(\log n)$) or very large (i.e., polynomial in $n$). In this paper, we provide answers for both regimes.

**Small Dimension.** Let us start with the small-dimension regime, $d = o(\log n)$. The outcome of the question whether better algorithms are possible is a priori not clear. Many related center problems (for which the goal is to select a center point $x^*$ that is closest not necessarily in the Hamming metric but in some other metric space) differ substantially in this regard: On the one hand, in the Euclidian metric, even for $d = 2^{O(\log^* n)}$, the center problem requires quadratic time under the Hitting Set Conjecture [1].[1] On the other hand, in stark contrast, the center problem for the $\ell_1$ and $\ell_\infty$ metrics can be solved in almost-linear time $n^{1+o(1)}$ whenever the dimension is $d = o(\log n)$. This dichotomy phenomenon extends to even more general problems including nearest and furthest neighbor questions for various metrics and the maximum inner product problem [33, 7].

In view of this, we obtain the perhaps surprising result that whenever $d = o(\log n)$ the discrete Closest String problem can indeed be solved in subquadratic – even almost-linear – time. More generally, we obtain the following algorithm:

▶ **Theorem 4** (Discrete Closest String for Small Dimensions). *The discrete Closest String problem can be solved in time $O(n \cdot 2^d)$.*

Note that this result is trivial for binary alphabets, and our contribution lies in finding an algorithm in time $O(n \cdot 2^d)$ for alphabets of *arbitrary* size.

We believe that this result is interesting also from a technical perspective, as it crucially relies on the *inclusion-exclusion* principle. While this technique is part of the everyday tool-set for exponential-time and parameterized algorithms, it is uncommon to find applications for polynomial-time problems and our algorithm yields the first such application to a center-type

[1] Technically, the problem is only known to be hard in the *listing* version where we require to list *all* feasible centers [1].

problem, to the best of our knowledge. We believe that our characterization of the Hamming distance in terms of an inclusion-exclusion-type formula (see Lemma 23) is very natural and likely to find applications in different contexts.

**Large Dimension.** In the large-dimension regime, where $d$ is polynomial in $n$, it is folklore that fast matrix multiplication should be of use. Specifically, over a binary alphabet we can solve the Closest String problem in time $O(\mathrm{MM}(n,d,n))$ (where $\mathrm{MM}(n,d,n)$ is the time to multiply an $n \times d$ by a $d \times n$ matrix) by using fast matrix multiplication to compute the Hamming distances between all pairs of vectors, rather than by brute-force. For arbitrary alphabet sizes this idea leads to a running time of $O(\mathrm{MM}(n,d|\Sigma|,n))$ which is of little use as $|\Sigma|$ can be as large as $n$ and in this case the running time becomes $\Omega(n^2 d)$.

We prove that nevertheless, the $O(n^2 d)$-time baseline algorithm can be improved using fast matrix multiplication – in fact, using ideas from *sparse* matrix multiplication such as Yuster and Zwick's heavy-light idea [34].

▶ **Theorem 5** (Discrete Closest String for Large Dimensions). *For all $\delta > 0$, there is some $\epsilon > 0$ such that the discrete Closest String problem with dimension $d = n^\delta$ can be solved in time $O(n^{2+\delta-\epsilon})$.*

**Remotest String.** Finally, we turn our attention to the discrete Remotest String problem. In light of the previously outlined equivalence in the continuous setting, we would expect that also in the discrete setting, the Closest and Remotest String problem are tightly connected. We confirm this suspicion and establish a strong equivalence for binary alphabets:

▶ **Theorem 6** (Equivalence of Discrete Closest and Remotest String). *If the discrete Closest String over a binary alphabet is in time $T(n,d)$, then the discrete Remotest String over a binary alphabet is in time $T(O(n), O(d+\log n)) + \widetilde{O}(nd)$. Conversely, if the discrete Remotest String over a binary alphabet is in time $T'(n,d)$, then the discrete Closest String over a binary alphabet is in time $T'(O(n), O(d+\log n)) + \widetilde{O}(nd)$.*

In combination with Theorem 3, this equivalence entails that also Remotest String requires quadratic time in the regime $\omega(\log n) < d < n^{o(1)}$. Let us remark that, while the analogous equivalence is trivial in the continuous regime, proving Theorem 6 is not trivial and involves the construction of a suitable gadget that capitalizes on explicit constant-weight codes.

The similarity between discrete Closest and Remotest String continues also on the positive side: All of our algorithms extend naturally to Remotest String, not only for binary alphabets (see the full version for more details).

## 1.3 Open Problems

Our work inspires some interesting open problems. The most pressing question from our perspective is whether there also is a $|\Sigma|^{(1-o(1))d}$ lower-bound for continuous Closest String (for alphabets of size bigger than 2).

▶ **Open Question 7** (Continuous Closest String for Large Alphabets). *For $|\Sigma| > 2$, can the continuous Closest String problem be solved in time $O(|\Sigma|^{(1-\epsilon)d}\,\mathrm{poly}(n))$, for some $\epsilon > 0$?*

We believe that our approach (proving hardness under SETH) hits a natural barrier for the Closest String problem. In some sense, the $k$-SAT problem behaves very similarly to Remotest String (with the goal to be remote from all falsifying assignments), and over binary alphabets remoteness and closeness can be exchanged. For larger alphabets this trivial equivalence simply does not hold. It would be exciting if this insight could fuel a faster *algorithm* for Closest String, and we leave this question for future work.

On the other hand, consider again the discrete Closest and Remotest String problems. While we close *almost all* regimes of parameters, there is one regime which we did not address in this paper:

▶ **Open Question 8** (Discrete Closest/Remotest String for Logarithmic Dimension). *Let $c$ be a constant. Can the discrete Closest and Remotest String problems with dimension $d = c \log n$ be solved in time $O(n^{2-\epsilon})$, for some $\epsilon = \epsilon(c) > 0$?*

In the regime $d = \Theta(\log n)$, we typically expect only very sophisticated algorithms, say using the polynomial method in algorithm design [2], to beat exhaustive search. And indeed, using the polynomial method it is possible to solve also discrete Closest and Remotest String in subquadratic time for binary (or more generally, constant-size) alphabets [5, 4, Theorem 1.4]. The question remains whether subquadratic time complexity is also possible for unrestricted alphabet sizes.

## 1.4 Outline of the Paper

We organize this paper as follows. In Section 2 we give some preliminaries and state the formal definitions of the continuous/discrete Closest/Remotest String problems. In Section 3 we prove our conditional hardness results for the continuous problems. In Section 4 we treat in detail the discrete problems. Throughout, due to space constraints, we defer several proofs to the full version of this paper.

# 2 Preliminaries

We set $[n] = \{1, \ldots, n\}$ and write $\widetilde{O}(T) = T(\log T)^{O(1)}$ and $\mathrm{poly}(n) = n^{o(1)}$. We occasionally write $\mathbf{1}(P) \in \{0, 1\}$ to express the truth value of the proposition $P$.

**Strings.** Let $\Sigma$ be a finite alphabet of size at least 2. For a string $x \in \Sigma^d$ of length (or dimension) $d$, we write $x[i]$ for the $i$-th character in $x$. For a subset $I \subseteq [d]$, we write $x[I] \in \Sigma^I$ for the subsequence obtained from $x$ by restricting to the characters in $I$. The *Hamming distance* between two equal-length strings $x, y \in \Sigma^d$ is defined as $\mathrm{HD}(x, y) = |\{i \in [d] : x[i] \neq y[i]\}|$. Let $X$ be a set of length-$d$ strings and let $x^*$ be a length-$d$ string. Then we set

$$r(x^*, X) = \max_{y \in X} \mathrm{HD}(x^*, y) \quad \text{(the } \textit{radius} \text{ of } X \text{ around } x^*\text{)},$$
$$d(x^*, X) = \min_{y \in X} \mathrm{HD}(x^*, y) \quad \text{(the } \textit{distance} \text{ from } x^* \text{ to } X\text{)}.$$

Let us formally repeat the definitions of the four problems studied in this paper:

▶ **Definition 9** (Continuous Closest String). *Given a set of $n$ strings $X \subseteq \Sigma^d$, find a string $x^* \in \Sigma^d$ which minimizes the radius $r(x^*, X)$.*

▶ **Definition 10** (Continuous Remotest String). *Given a set of $n$ strings $X \subseteq \Sigma^d$, find a string $x^* \in \Sigma^d$ which maximizes the distance $d(x^*, X)$.*

▶ **Definition 11** (Discrete Closest String). *Given a set of $n$ strings $X \subseteq \Sigma^d$, find a string $x^* \in X$ which minimizes the radius $r(x^*, X)$.*

▶ **Definition 12** (Discrete Remotest String). *Given a set of $n$ strings $X \subseteq \Sigma^d$, find a string $x^* \in X$ which maximizes the distance $d(x^*, X \setminus \{x^*\})$.*

**Hardness Assumptions.** In this paper, our lower bounds are conditioned on the following two plausible hypotheses from fine-grained complexity.

▶ **Definition 13** (Strong Exponential Time Hypothesis, SETH [18, 19]). *For all $\epsilon > 0$, there is some $k \geq 1$ such that $k$-CNF SAT cannot be solved in time $O(2^{(1-\epsilon)n})$.*

▶ **Definition 14** (Hitting Set Conjecture [3]). *For all $\epsilon > 0$, there is some $c \geq 1$ such that no algorithm can decide in $O(n^{2-\epsilon})$ time, whether in two given lists $A, B$ of $n$ subsets of a universe of size $c \log n$, there is a set in the first list that intersects every set in the second list (i.e. a "hitting set").*

## 3 Continuous Closest String is SETH-Hard

In this section we present our fine-grained lower bounds for the continuous Closest and Remotest String problems. We start with a high-level overview of our proof, and then provide the technical details in Sections 3.1–3.4.

Let us first recall that over *binary* alphabets, the continuous Closest and Remotest String problems are trivially equivalent. The insight is that for any two strings $x, y \in \{0,1\}^d$ we have that $\mathrm{HD}(x, y) = d - \mathrm{HD}(\overline{x}, y)$ where $\overline{x}$ is the *complement* of $x$ obtained by flipping each bit. From this it easily follows that

$$\min_{x^* \in \{0,1\}^d} \max_{y \in X} \mathrm{HD}(x^*, y) = d - \max_{x^* \in \{0,1\}^d} \min_{y \in X} \mathrm{HD}(x^*, y).$$

Note that finding a string $x^*$ optimizing the left-hand side is exactly the Closest String problem, whereas finding a string $x^*$ optimizing the right-hand side is exactly the Remotest String problem, and thus both problems are one and the same. For this reason, let us focus our attention for the rest of this section only on the Remotest String problem.

**Tight Lower Bound for Remotest String.** Our goal is to establish a lower bound under the Strong Exponential Time Hypothesis. To this end, we reduce a $k$-SAT instance with $N$ variables to an instance of the Remotest String problem with dimension $d = (1 + o(1))N$. In Sections 3.1–3.4 we will actually reduce from a $q$-ary analogue of the $k$-SAT in order to get a tight lower bound for all alphabet sizes $|\Sigma|$. However, for the sake of simplicity we stick to binary strings and the usual $k$-SAT problem in this overview. Our reduction runs in two steps.

**Step 1: Massaging the SAT Formula.** In the first step, we bring the given SAT formula into a suitable shape for the reduction to the Remotest String problem. Throughout, we partition the variables $[N]$ into *groups* $P_1, \ldots, P_{\frac{N}{s}}$ of size exactly $s$ (where $s$ is a parameter to be determined later). We assert the following properties:

- *Regularity:* All clauses contain exactly $k$ literals, and all clauses contain literals from the same number of groups (say $r$). This property can be easily be guaranteed by adding a few fresh variables to the formula, all of which must be set to 0 in a satisfying assignment, and by adding these variables to all clauses which do not satisfy the regularity constraint yet.
- *Balancedness:* Let us call an assignment $\alpha \in \{0,1\}^N$ *balanced* if in every group it assigns exactly half the variables to 0 and half the variables to 1. We say that a formula is *balanced* if it is either unsatisfiable or if it is satisfiable by a balanced assignment. To make sure that a given formula is balanced, we can for instance flip each variable in the

formula with probability $\frac{1}{2}$. In this way we balance each group with probability $\approx \frac{1}{\sqrt{s}}$, and so all $\frac{N}{s}$ groups are balanced with probability at least $s^{-\frac{N}{2s}}$. By choosing $s = \omega(1)$, this random experiment yields a balanced formula after a negligible number of repetitions. In Lemma 19 we present a deterministic implementation of this idea.

**Step 2: Reduction to Remotest String.** The next step is to reduce a regular and balanced $k$-CNF formula to an instance of the Remotest String problem. The idea is to encode all *falsifying* assignments of the formula as strings – a sufficiently remote point should in spirit be remote from falsifying and thus satisfying. To implement this idea, take any clause $C$ from the instance. Exploiting the natural correspondence between strings and assignments, we add all strings $\alpha \in \{0,1\}^n$ that satisfy the following two constraints to the Remotest String instance:

1. The assignment $\alpha$ falsifies the clause $C$.
2. For any group $P_i$ that does *not* contain a variable from $C$, we have that $\alpha[P_i] = 0^s$ or $\alpha[P_i] = 1^s$.

We start with the intuition behind the second constraint: For any *balanced* assignment $\alpha$ and any group $P_i$ that does not contain a variable from $C$, we have that $\mathrm{HD}(\alpha^*[P_i], \alpha[P_i]) = \frac{s}{2}$ (the string $\alpha^*[P_i]$ contains half zeros and half ones, whereas $\alpha[P_i]$ is either all-zeros or all-ones). There are exactly $\frac{N}{s} - r$ such groups (by the regularity), leading to Hamming distance $\frac{s}{2}(\frac{N}{s} - r)$.

It follows that the only groups that actually matter for the distance between $\alpha^*$ and $\alpha$ are the groups which do contain a variable from $C$. Here comes the first constraint into play: If $\alpha^*$ is a satisfying assignment, then $\alpha^*$ and $\alpha$ must differ in at least one of these groups and therefore have total distance at least $\frac{s}{2}(\frac{N}{s} - r) + 1$. Conversely, for any falsifying assignment $\alpha^*$ there is some string $\alpha$ in the instance with distance at most $\frac{s}{2}(\frac{N}{s} - r)$. Therefore, to decide whether the SAT formula is satisfiable it suffices to compute whether there is a Remotest String with distance at least $\frac{s}{2}(\frac{N}{s} - r) + 1$. Finally, it can be checked that the number of strings $\alpha$ added to the instance is manageable.

This completes the outline of our hardness proof, and we continue with the details. In Section 3.1 we introduce the $(q,k)$-SAT problem which we will use to give a clean reduction also for alphabet larger than size 2. In Section 3.2 we formally prove how to guarantee that a given $(q,k)$-SAT formula is regular and balanced, and in Section 3.3 we give the details about the reduction to the Remotest String problem. We put these pieces together in Section 3.4 and formally prove Theorem 2.

## 3.1 $q$-ary SAT

To obtain our full hardness result, we base our reduction on the hardness of *$q$-ary* analogue of the classical $k$-SAT problem. We start with an elaborate definition of this problem. Let $X_1, \ldots, X_N$ denote some $q$-ary variables (i.e., variables taking values in the domain $[q]$). A *literal* is a Boolean predicate of the form $X_i \neq a$, where $x_i$ is one of the variables and $a \in [q]$. A *clause* is a disjunction of several literals; we say the clause has *width* $k$ if it contains exactly $k$ literals. A *$(q,k)$-CNF formula* is a disjunction of clauses of width at most $k$. Finally, in the $(q,k)$-SAT problem, we are given a $(q,k)$-CNF formula over $M$ clauses and $N$ $q$-ary variables, and the task is to check whether there exists an assignment $\alpha \in [q]^N$ which satisfies all clauses. This problem has already been addressed in previous works [31, 30], and it is known that $q$-ary SAT cannot be solved faster than exhaustive search unless SETH fails:

▶ **Lemma 15** ($q$-ary SAT is SETH-Hard [30, Theorem 3.3]). *For any $\epsilon > 0$, there is some $k \geq 3$ such that for all $q = q(N) \geq 2$, $(q,k)$-SAT cannot be solved in time $O(q^{(1-\epsilon)N}\operatorname{poly}(M))$, unless SETH fails.*

While $k$ is always constant, note that this hardness result applies even when $q$ grows with $N$. We will later exploit this by proving hardness for Remotest String even for alphabets of super-constant size.

## 3.2 Regularizing and Balancing

Before we get to the core of our hardness result, we need some preliminary lemmas on the structure of $(q,k)$-CNF formulas. Throughout, let $N$ be the number of variables and let $\mathcal{P}$ be a partition of $N$ into *groups* of size exactly $s$. (Note that the existence of $P$ implies that $N$ is divisible by $s$.) In two steps we will now formally introduce the definitions of regular and balanced formulas and show how to convert unconstrained formulas into regular and balanced ones. We defer the proofs of the upcoming lemmas to the full version of this paper.

▶ **Definition 16** (Regular Formulas). *Let $\phi$ be a $(q,k)$-CNF formula over $N$ variables, and let $\mathcal{P}$ be a partition of $[N]$. We say that $\phi$ is* $r$-regular *(with respect to $\mathcal{P}$) if every clause contains exactly $k$ literals from exactly $r$ distinct groups in $\mathcal{P}$.*

▶ **Lemma 17** (Regularizing). *Let $\phi$ be a $(q,k)$-CNF formula, and let $2k \leq s \leq N$. In time $\operatorname{poly}(NM)$ we can construct a $(q,2k)$-CNF formula $\phi'$ satisfying the following properties:*

- *$\phi'$ is satisfiable if and only if $\phi$ is satisfiable.*
- *$\phi'$ has at most $N + O(s)$ variables and at most $M + O(s\operatorname{poly}(q))$ clauses.*
- *$\phi'$ is $(k+1)$-regular with respect to some partition $\mathcal{P}$ into groups of size exactly $s$.*

▶ **Definition 18** (Balanced Formulas). *Let $\mathcal{P}$ be a partition of $[N]$ into groups of size $s$. We say that an assignment $\alpha \in [q]^N$ is* balanced *(with respect to $\mathcal{P}$) if in every group of $\mathcal{P}$, $\alpha$ assigns each symbol in $[q]$ exactly $\frac{s}{q}$ times. We say that a $(q,k)$-CNF formula $\phi$ is* balanced *(with respect to $\mathcal{P}$) if either $\phi$ is unsatisfiable, or $\phi$ is satisfiable by a balanced assignment $\alpha$.*

▶ **Lemma 19** (Balancing). *Let $\phi$ be a $(q,k)$-CNF formula over $N$ variables, let $\mathcal{P}$ be a partition of $[N]$ into groups of size $s$, and assume that $q$ divides $s$. We can construct $(q,k)$-CNF formulas $\phi_1, \ldots, \phi_t$ over the same number of variables and clauses as $\phi$ such that:*

- *For all $i \in [t]$, $\phi_i$ is satisfiable if and only if $\phi$ is satisfiable.*
- *There is some $i \in [t]$ such that $\phi_i$ is balanced (with respect to $\mathcal{P}$).*
- *$t = ((s+1)(q-1))^{(q-1)\frac{N}{s}}$, and we can construct each formula in time $\operatorname{poly}(NMt)$.*

## 3.3 Reduction to Remotest String

Having in mind that for our reduction we can assume the SAT formula to be regular and balanced, the following lemma constitutes the core of our reduction:

▶ **Lemma 20** (Reduction from Regular Balanced SAT to Remotest String). *Suppose there is an algorithm for the continuous Remotest String problem, running in time $O(|\Sigma|^{(1-\epsilon)d}\operatorname{poly}(n))$, for some $\epsilon > 0$. Then there is an algorithm that decides whether a given $s$-partitioned $r$-regular $(q,k)$-SAT formula is satisfiable, and runs in time $O(q^{(1-\epsilon)N + O(s+\frac{N}{s})}\operatorname{poly}(M))$.*

**Proof.** We start with some notation: For a clause $C$, we write $\mathcal{P}(C) \subseteq \mathcal{P}$ to address all groups containing a literal from $C$. We start with the construction of the Remotest String instance with alphabet $\Sigma = [q]$ and dimension $d = N$. Here, we make use of the natural

correspondence between strings $\alpha \in \Sigma^d$ and assignments $\alpha \in [q]^N$. In the instance, we add the following strings: For each clause $C$, add all assignments $\alpha \in [q]^N$ to the instance which satisfy the following two constraints:

1. The assignment $\alpha$ falsifies the clause $C$.
2. For each group $P \in \mathcal{P} \setminus \mathcal{P}(C)$, the subsequence $\alpha[P]$ contains only one symbol. (That is, $\alpha[P] = a^s$ for some $a \in [q]$.)

We prove that this instance is complete and sound.

▷ Claim 21 (Completeness). If $\phi$ is satisfiable, then there is some $\alpha^* \in [q]^d$ with $d(\alpha^*, X) > \frac{(q-1)(N-rs)}{q}$.

Proof. Since we assume that the formula $\phi$ is satisfiable and balanced, there is a satisfying and balanced assignment $\alpha^*$. To prove that $d(\alpha^*, X) > \frac{(q-1)(N-rs)}{q}$, we prove that for each string $\alpha$ added to the Remotest String instance, we have $\mathrm{HD}(\alpha^*, \alpha) > \frac{(q-1)(N-rs)}{q}$ . Let $C$ be the clause associated to $\alpha$. From the two conditions on $\alpha$, we get the following two bounds.

By the first condition, $\alpha$ is a falsifying assignment of $C$. In particular, the subsequence $\alpha[\bigcup_{P\in\mathcal{P}(C)} P]$ falsifies $C$ (which is guaranteed to contain all variables visible to $C$) falsifies $C$. Since $\alpha^*$ is a satisfying assignment to the whole formula, and in particular to $C$, we must have that $\alpha^*[\bigcup_{P\in\mathcal{P}(C)} P] \neq \alpha[\bigcup_{P\in\mathcal{P}(C)} P]$, and thus $\sum_{P\in\mathcal{P}(C)} \mathrm{HD}(\alpha^*[P], \alpha[P]) \geq 1$.

By the second condition, for any group $P \in \mathcal{P} \setminus \mathcal{P}(C)$, the subsequence $\alpha[P]$ contains only one symbol. Since $\alpha^*$ is balanced, $\alpha^*[P]$ contains that symbol exactly in a $1/q$-fraction of the positions and differs in the remaining ones from $\alpha[P]$. It follows that $\mathrm{HD}(\alpha^*[P], \alpha[P]) = s - \frac{s}{q} = \frac{(q-1)s}{q}$.

Combining both bounds, we have that

$$\begin{aligned}\mathrm{HD}(\alpha^*, \alpha) &= \sum_{P\in\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) + \sum_{P\in\mathcal{P}\setminus\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) \\ &\geq 1 + \left(\frac{N}{s} - r\right) \cdot \frac{(q-1)s}{q} = \frac{(q-1)(N-rs)}{q} + 1,\end{aligned}$$

and the claim follows. ◁

▷ Claim 22 (Soundness). If $\phi$ is not satisfiable, then for all $\alpha^* \in [q]^d$ we have $d(\alpha^*, X) \leq \frac{(q-1)(N-rs)}{q}$.

Proof. Take any $\alpha^* \in [q]^d$. Since $\phi$ is not satisfiable, $\alpha^*$ is a falsifying assignment of $\phi$ and thus there is some clause $C$ that is falsified by $\alpha^*$. Our strategy is to find some string $\alpha \in [q]^d$ in the constructed instance with $\mathrm{HD}(\alpha^*, \alpha) \leq \frac{(q-1)(N-rs)}{q}$.

We define that string $\alpha$ group-wise: In the groups $\mathcal{P}(C)$ touching $C$, we define $\alpha$ to be exactly as $\alpha^*$, that is, $\alpha[\bigcup_{P\in\mathcal{P}(C)}] := \alpha^*[\bigcup_{P\in\mathcal{P}(C)}]$. For each group $P \in \mathcal{P} \setminus \mathcal{P}(C)$ not touching $C$, let $a \in [q]$ be an arbitrary symbol occurring at least $\frac{s}{q}$ times in $\alpha^*[P]$ and assign $\alpha[P] := a^s$. By this construction we immediately have $\mathrm{HD}(\alpha[P], \alpha^*[P]) \leq s - \frac{s}{q} = \frac{(q-1)s}{q}$, and in total

$$\begin{aligned}\mathrm{HD}(\alpha^*, \alpha) &= \sum_{P\in\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) + \sum_{P\in\mathcal{P}\setminus\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) \\ &\leq 0 + \left(\frac{N}{s} - r\right) \cdot \frac{(q-1)s}{q} = \frac{(q-1)(N-rs)}{q},\end{aligned}$$

as claimed. ◁

In combination, Claims 21 and 22 show that the constructed instance of the Remotest String problem is indeed equivalent to the given $(q,k)$-SAT instance $\phi$ in the sense that $\phi$ is satisfiable if and only if there is a remote string with distance more than $\frac{(q-1)(N-rs)}{q}$.

It remains to analyze the running time. Let $n$ denote the number of strings in the constructed instance. As a first step, we prove that $n \leq q^{O(s)+\frac{N}{s}} \cdot M$ and that we can construct the instance in time $\text{poly}(n)$. Indeed, focus on any clause $C$. The strings $\alpha$ in the instance are unconstrained in all groups touching $C$ (up to the condition that $\alpha$ must falsify $C$) which accounts for $r \cdot s$ positions and thus $q^{rs} = q^{O(s)}$ options. For each group not touching $C$ we can choose between $q$ possible values, and therefore the total number of options is $q^{\frac{N}{s}-r} \leq q^{\frac{N}{s}}$. Therefore, the total number of strings is indeed $n \leq M \cdot q^{O(s)} \cdot q^{\frac{N}{s}}$. Moreover, it is easy to see that the instance can be constructed in time $\text{poly}(n)$.

As the time to construct the instance is negligible, the total running time is dominated by solving the Remotest String instance. Assuming an algorithm in time $O(|\Sigma|^{(1-\epsilon)d}\,\text{poly}(n))$, this takes time $O(q^{(1-\epsilon)N+O(s+\frac{N}{s})}\,\text{poly}(M))$ as claimed. ◀

## 3.4 Putting the Pieces Together

We are finally ready to prove Theorems 1 and 2.

▶ **Theorem 2** (Continuous Remotest String is SETH-Hard). *The continuous Remotest String problem cannot be solved in time $O(|\Sigma|^{(1-\epsilon)d}\,\text{poly}(n))$, for any $\epsilon > 0$ and $|\Sigma| = o(d)$, unless SETH fails.*

**Proof.** Suppose that the continuous Remotest String problem is in time $O(|\Sigma|^{(1-\epsilon)d}\,\text{poly}(n))$ for some $\epsilon > 0$ and for $|\Sigma| = o(d)$. With this in mind, we design a better-than-brute-force $(q,k)$-SAT algorithm for $q = |\Sigma|$ by combining the previous three Lemmas 17, 19, and 20. Let $\phi$ be the input formula, and let $\mathcal{P}$ denote a partition of the variables into groups of size $s$ (which is yet to be determined) as before.

1. Using Lemma 17, construct a *regular* $(q,2k)$-formula $\phi'$ which is equivalent to $\phi$.
2. Using Lemma 19, construct regular $(q,2k)$-formulas $\phi'_1,\ldots,\phi'_t$ all of which are equivalent to $\phi$. At least one of these formulas is *balanced*.
3. By means of the reduction in Lemma 20, solve all $t$ formulas $\phi'_1,\ldots,\phi'_t$. If a formula is reported to be satisfiable, check whether the answer is truthful (e.g., using the standard decision-to-reporting reduction) and if so report that the formula is satisfiable. We need the additional test since, strictly speaking, we have not verified in Lemma 20 that the algorithm is correct for non-balanced inputs.

The correctness is obvious. Let us analyze the running time. Constructing the formula $\phi'$ takes polynomial time and can be neglected. By Lemma 17, $\phi'$ has $N' = N + O(s)$ variables and $M' = M + O(s\,\text{poly}(q))$ clauses. The construction of the formulas $\phi'_1,\ldots,\phi'_t$ also runs in polynomial time $\text{poly}(N'M't)$ and can be neglected; this time we do not increase the number of variables and clauses. Moreover, Lemma 19 guarantees that

$$t = ((s+1)(q-1))^{(q-1)\frac{N'}{s}} \leq (sq)^{O(\frac{qN}{s})},$$

By picking $s = cq$ (for some parameter $c$ to be determined), this becomes

$$t \leq (cq^2)^{O(\frac{N}{c})} = q^{O(\frac{N}{c}\log_q(cq^2))} = q^{N\cdot O(\frac{\log c}{c})}.$$

Finally, by Lemma 20 solving each formula $\phi'_i$ takes time

$$q^{(1-\epsilon)N'+O(s+\frac{N'}{s})}\,\text{poly}(M') = q^{(1-\epsilon)N+O(s+\frac{N}{s})}\,\text{poly}(M) = q^{(1-\epsilon)N+o(cN)+O(\frac{N}{c})}\,\text{poly}(M),$$

(using that $s = cq = o(cN)$), and thus the total running time is bounded by

$$q^{N \cdot O(\frac{\log c}{c})} \cdot q^{(1-\epsilon)N + o(cN) + O(\frac{N}{c})} \operatorname{poly}(M) = q^{(1-\epsilon+o(c)+O(\frac{\log c}{c}))N} \operatorname{poly}(M).$$

Note that by picking $c$ to be a sufficiently large constant (depending on $\epsilon$), the exponent becomes $(1 - \frac{\epsilon}{2})N$, say. We have therefore obtained an algorithm for the $(q, k)$-SAT problem in time $O(q^{(1-\epsilon/2)N} \operatorname{poly}(M))$, which contradicts SETH by Lemma 15. ◀

## 4 Discrete Closest String via Inclusion-Exclusion

In this section, we present an algorithm for the discrete Closest String problem with *subquadratic* running time whenever the dimension is small, i.e. $d = o(\log n)$. Our algorithm relies on the inclusion-exclusion principle, and is, to the best of our knowledge, the first application of this technique to the Closest and Remotest String problems. Specifically, we obtain the following result:

▶ **Theorem 4** (Discrete Closest String for Small Dimensions). *The discrete Closest String problem can be solved in time $O(n \cdot 2^d)$.*

We structure this section as follows: First, we present a high-level overview of the main ideas behind the algorithm; for the sake of presentation, we focus only on the Closest String problem. We start developing a combinatorial toolkit to tackle the Closest String problem (with all proofs deferred to the full version of this paper). Then, in Section 4.1 we provide the actual algorithm and prove Theorem 4.

Before we describe our algorithm, we provide some intuition about the general connection between the inclusion-exclusion principle and the Hamming distance between a pair of strings. Our key insight is that the inclusion-exclusion principle allows us to express whether two strings have Hamming distance bounded by, say $k$. The following lemma makes this idea precise:

▶ **Lemma 23** (Hamming Distance by Inclusion-Exclusion). *Let $x$ and $y$ be two strings of length $d$ over some alphabet $\Sigma$, and let $0 \leq k < d$. Then:*

$$\mathbf{1}(\mathrm{HD}(x, y) \leq k) = \sum_{\substack{I \subseteq [d] \\ |I| \geq d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot \mathbf{1}(x[I] = y[I]).$$

Recall that we write $x[I] = y[I]$ to express that the strings $x$ and $y$ are equally restricted to the indices in $I$. The precise inclusion-exclusion-type formula does not matter too much here, but we provide some intuition for Lemma 23 by considering the special cases where $\mathrm{HD}(x, y) = k$ and $\mathrm{HD}(x, y) = k - 1$. If $\mathrm{HD}(x, y) = k$, then there is a unique set $I$ of size $d - k$ for which $x[I] = y[I]$. If instead $\mathrm{HD}(x, y) = k - 1$, then there is a unique such set of size $d - k + 1$, and additionally there are $d - k + 1$ such sets of size $d - k$. The scalars $(-1)^{|I|-d+k}\binom{|I|-1}{d-k-1}$ are chosen in such a way that in any case, all these contributions sum up to exactly 1.

The takeaway from the above lemma is that we can express the proposition that two strings satisfy $\mathrm{HD}(x, y) \leq k$ by a linear combination of $2^d$ indicators of the form $\mathbf{1}(x[I] = y[I])$. It is easy to extend this idea further to the following lemma, which is the core of our combinatorial approach:

▶ **Lemma 24** (Radius by Inclusion-Exclusion). *Let $x$ be a string of length $d$ over some alphabet $\Sigma$, let $X$ be a set of strings each of length $d$ over $\Sigma$, and let $0 \leq k < d$. Then $r(x, X) \leq k$ if and only if*

$$|X| = \sum_{\substack{I \subseteq [d] \\ |I| \geq d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|.$$

Given this lemma, our algorithm for the Closest String problem is easy to state. Informally, we proceed in the following two steps:

**Step 1: Partition.** Precompute, for all $x \in X$ and for all $I \subseteq [d]$, the value $|\{y \in X : x[I] = y[I]\}|$. This can be implemented in time $O(n \cdot 2^d \cdot \text{poly}(d))$ by partitioning the input strings $X$ depending on their characters in the range $I$. After computing this partition, we can read the value $|\{y \in X : x[I] = y[I]\}|$ as the number of strings in the same part as $x$.

**Step 2: Inclusion-Exclusion.** We test for each $0 \leq k \leq d$ and $x \in X$, whether $r(x, X) \leq k$ and finally return the best answer. By Lemma 24 we can equivalently express the event $r(x, X) \leq k$ via

$$|X| = \sum_{\substack{I \subseteq [d] \\ |I| \geq d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|.$$

By observing that the sum contains only $2^d$ terms and noting that we have precomputed the values $|\{y \in X : x[I] = y[I]\}|$, we can evaluate the sum, for a fixed $x$, in time $O(2^d \cdot \text{poly}(d))$. In total, across all strings $x \in X$, the running time becomes $O(n \cdot 2^d \cdot \text{poly}(d))$.

Finally, let us briefly comment on the $\text{poly}(d)$ term in the running time. When evaluating the above sum naively, we naturally incur a running time overhead of $\text{poly}(d)$ since the numbers in the sum need $\Omega(d + \log n)$ bits to be represented. However, this overhead can be circumvented by evaluating the expression in a smarter way. We provide more details in Section 4.1.

## 4.1 The Algorithm in Detail

In this subsection, we provide our algorithms for the discrete Closest String problem. Let us first demonstrate how to precompute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ efficiently.

▶ **Lemma 25.** *We can compute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ in time $O(n \cdot 2^d)$.*

**Proof.** Our strategy is to compute, for each $I \subseteq [d]$, a partition $\mathcal{P}_I$ of the set of all strings $X$ such that two strings $y_1, y_2 \in X$ are in the same part in $\mathcal{P}_I$ if and only if $y_1[I] = y_2[I]$. This is our goal since, for all strings $x \in X$, the value we are interested in $|\{y \in X : x[I] = y[I]\}|$ is exactly the size of the part $P$ in $\mathcal{P}_I$ that contains $x$. Thus, if we can efficiently compute, for all $I \subseteq [d]$ and all $x \in X$, the partition $\mathcal{P}_I$ and the part $P \in \mathcal{P}_I$ such that $x \in P$ then we have the desired algorithm.

Computing the partition $\mathcal{P}_I$ for each subset of $I \subseteq [d]$ when $|I| \leq 1$ is simple: The partition $\mathcal{P}_\emptyset$ contains just one part which is the entire input set. We also know that $\mathcal{P}_{\{i\}} = \{\{x \in X : x[i] = \sigma\} : \sigma \in \Sigma\}$ for every $0 \leq i \leq d-1$. Thus, we can compute the partitions $\mathcal{P}_\emptyset$ and $\mathcal{P}_{\{i\}}$ for every $0 \leq i \leq d-1$ in time $O(n \cdot d)$. The remaining question is how to efficiently compute the partitions $\mathcal{P}_I$ for each subset of $I \subseteq [d]$ where $|I| \geq 2$.

The idea is to use dynamic programming in combination with a *partition refinement* data structure. Let us start with some notation: For a partition $\mathcal{P}$ and a set $S$, we define the *refinement* of $\mathcal{P}$ by $S$ as the partition $\{P \cap S, P \setminus S : P \in \mathcal{P}\}$. For two partitions $\mathcal{P}$ and $\mathcal{P}'$, we define the refinement of $\mathcal{P}$ by $\mathcal{P}'$ by the iterative refinement of all sets $S \in \mathcal{P}'$. In previous work, Habib, Paul, and Viennot [16] have established a data structure to maintain partitions $\mathcal{P}$ of some universe $[n]$ that efficiently supports the following two operations:

- *Refinement:* We can refine a partition $\mathcal{P}$ by another partition $\mathcal{P}'$ in time $O(n)$.
- *Query:* Given a partition $\mathcal{P}$ and an element $i \in [d]$, we can find the part $i \in P \in \mathcal{P}$ in time $O(1)$.

Given this data structure, our algorithm is simple: Enumerate all sets $I$ in nondecreasing order with respect to their sizes $|I|$. Writing $I = I' \cup \{i\}$ (for some $i \in [d]$), we compute $\mathcal{P}_I$ as the refinement of the previously computed partitions $\mathcal{P}_{I'}$ and $\mathcal{P}_{\{i\}}$. It is straightforward to verify that this algorithm is correct. The running time of each refinement step is $O(n)$ and so the total running time is $O(n \cdot 2^d)$ as claimed. ◀

We are finally ready to state our algorithm and prove its correctness using Lemmas 24 and 25.

**Proof of Theorem 4.** First, it is clear that if we test for each $0 \leq k \leq d$ and $x \in X$ whether $r(x, X) \leq k$ then we can find the solution to the discrete Closest String problem. From Lemma 24 we know that $r(x, X) \leq k$ if and only if:

$$|X| = \sum_{\substack{I \subseteq [d] \\ |I| \geq d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|.$$

Thus, if we efficiently compute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ and efficiently compute the right-hand side of the equation we will have an efficient algorithm for the discrete Closest String problem. We know from Lemma 25 that we can precompute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ in time $O(n \cdot 2^d)$. Therefore, the only missing part of the algorithm is computing the inclusion-exclusion step in $O(n \cdot 2^d)$ time.

If we naively evaluate the inclusion-exclusion formula the running time becomes $\Omega(n \cdot 2^d \cdot d)$ as the intermediate values need $\Omega(d)$ bits to be represented in memory. However, we observe that inclusion-exclusion formula can indeed be evaluated more efficiently by rewriting it as follows:

$$\sum_{\substack{I \subseteq [d] \\ |I| \geq d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|$$

$$= \sum_{\ell=d-k}^{d} (-1)^{\ell} \cdot \binom{\ell-1}{d-k-1} \cdot \sum_{\substack{I \subseteq [d] \\ |I| = \ell}} |\{y \in X : x[I] = y[I]\}|.$$

We can precompute $S[x, \ell] := \sum_{I \subseteq [d], |I| = \ell} |\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ and all values $1 \leq \ell \leq d$ before we compute the inclusion exclusion step. Since there are $2^d$ different subsets of $[d]$ and since we already have access to the values $|\{y \in X : x[I] = y[I]\}|$, for all strings $x \in X$, computing $S[x, \ell]$ amounts to time $O(n \cdot 2^d)$. Afterwards, computing

$$\sum_{\ell=d-k}^{d} (-1)^{\ell} \cdot \binom{\ell-1}{d-k-1} \cdot S[x, \ell]$$

for all strings $x \in X$ and for all $0 \leq k \leq d-1$ only takes time $O(n \cdot d^3)$. Hence, the total running time of the algorithm is $O(n \cdot 2^d)$. ◀

**Algorithm 1** An algorithm for the discrete Closest String problem in the small-distance regime. See Theorem 4.

*(Step 1: Precompute $T[x, I] = |\{y \in X : x[I] = y[I]\}|$)*
$\mathcal{P}_\emptyset \leftarrow X$
$\mathcal{P}_{\{i\}} \leftarrow \{\{x \in X : x[i] = \sigma\} : \sigma \in \Sigma\} \ \forall i \in [0, \ldots, d-1]$
**for** $I = I' \cup \{i\}$ **do**
    $\mathcal{P}_I \leftarrow$ refinement of $\mathcal{P}_{I'}, \mathcal{P}_{\{i\}}$
**for** $x \in X, I \subseteq [d]$ **do**
    $T[x, I] \leftarrow |P|$ where $x \in P \in \mathcal{P}_I$

*(Step 2: Inclusion-Exclusion)*
**for** $x \in X, I \subseteq [d]$ **do**
    $S[x, |I|] \leftarrow S[x, |I|] + T[x, I]$
**for** $k \leftarrow 0, \ldots, d-1$ **do**
    **for** $x \in X$ **do**
        **if** $|X| = \sum_{\ell=d-k}^{d} (-1)^\ell \cdot \binom{\ell-1}{d-k-1} \cdot S[x, \ell]$ **then**
            return $x$
return an arbitrary $x \in X$

We summarize the pseudocode of the algorithm outlined in the proof of Theorem 4 in Algorithm 1.

## References


1 Amir Abboud, MohammadHossein Bateni, Vincent Cohen-Addad, Karthik C. S., and Saeed Seddighin. On complexity of 1-center in various metrics. *CoRR*, abs/2112.03222, 2021. arXiv:2112.03222.

2 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

4 Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 467–476. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.57.

5 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 136–150. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.18.

6 Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2009. doi:10.1007/978-3-642-03685-9_26.

7 Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. *Theory Comput.*, 16:1–50, 2020. doi:10.4086/toc.2020.v016a004.

8 Gérard Cohen, Iiro Honkala, Simon Litsyn, and Antoine Lobstein. *Covering Codes*. ISSN. Elsevier Science, 1997.

9 Cláudio Nogueira de Meneses, Zhaosong Lu, Carlos A. S. Oliveira, and Panos M. Pardalos. Optimal solutions for the closest-string problem via integer programming. *INFORMS J. Comput.*, 16(4):419–429, 2004. doi:10.1287/ijoc.1040.0090.

10 Joaquín Dopazo, A. Rodriguez, J. C. Saiz, and Francisco Sobrino. Design of primers for PCR amplification of highly variable genomes. *Comput. Appl. Biosci.*, 9(2):123–125, 1993. doi:10.1093/bioinformatics/9.2.123.

11 Moti Frances and Ami Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997. doi:10.1007/s002240000044.

12 Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Efficient approximation algorithms for the hamming center problem. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 905–906. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.315081.

13 Jens Gramm, Falk Huner, and Rolf Niedermeier. Closest strings, primer design, and motif search. In *Sixth Annual International Conference on Computational Molecular Biology*, June 2002.

14 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003. doi:10.1007/s00453-003-1028-3.

15 Venkatesan Guruswami, Daniele Micciancio, and Oded Regev. The complexity of the covering radius problem on lattices and codes. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 161–173. IEEE Computer Society, 2004. doi:10.1109/CCC.2004.1313831.

16 Michel Habib, Christophe Paul, and Laurent Viennot. A synthesis on partition refinement: A useful routine for strings, graphs, boolean matrices and automata. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 1998. doi:10.1007/BFb0028546.

17 Ishay Haviv and Oded Regev. Hardness of the covering radius problem on lattices. *Chic. J. Theor. Comput. Sci.*, 2012, 2012. URL: http://cjtcs.cs.uchicago.edu/articles/2012/4/contents.html.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

20 Yuval Kochman, Arya Mazumdar, and Yury Polyanskiy. The adversarial joint source-channel problem. In *Proceedings of the 2012 IEEE International Symposium on Information Theory, ISIT 2012, Cambridge, MA, USA, July 1-6, 2012*, pages 2112–2116. IEEE, 2012. doi:10.1109/ISIT.2012.6283735.

21 J. Kevin Lanctôt. *Some String Problems in Computational Biology*. PhD thesis, University of Waterloo, 2004.

22 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Inf. Comput.*, 185(1):41–55, 2003. doi:10.1016/S0890-5401(03)00057-9.

23 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002. doi:10.1145/506147.506150.

24 K. Lucas, M. Busch, S. Mossinger, and J. A. Thompson. An improved microcomputer program for finding gene- or gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. *Comput. Appl. Biosci.*, 7(4):525–529, 1991. doi:10.1093/bioinformatics/7.4.525.

25 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009. doi:10.1137/080739069.

26 Holger Mauch, Michael J. Melzer, and John S. Hu. Genetic algorithm approach for the closest string problem. In *2nd IEEE Computer Society Bioinformatics Conference, CSB 2003, Stanford, CA, USA, August 11-14, 2003*, pages 560–561. IEEE Computer Society, 2003. doi:10.1109/CSB.2003.1227407.

27 Arya Mazumdar, Yury Polyanskiy, and Barna Saha. On chebyshev radius of a set in hamming space and the closest string problem. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 1401–1405. IEEE, 2013. doi:10.1109/ISIT.2013.6620457.

28 Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to ajtai's connection factor. *SIAM J. Comput.*, 34(1):118–169, 2004. doi:10.1137/S0097539703433511.

29 V. Proutski and Edward C. Holmes. Primer master: a new program for the design and analysis of PCR primers. *Comput. Appl. Biosci.*, 12(3):253–255, 1996. doi:10.1093/bioinformatics/12.3.253.

30 Noah Stephens-Davidowitz and Vinod Vaikuntanathan. Seth-hardness of coding problems. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 287–301. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00027.

31 Patrick Traxler. The time complexity of constraint satisfaction. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2008. doi:10.1007/978-3-540-79723-4_18.

32 Ying Wang, Wei Chen, Xu Li, and Bing Cheng. Degenerated primer design to amplify the heavy chain variable region from immunoglobulin cdna. *BMC Bioinform.*, 7(S-4), 2006. doi:10.1186/1471-2105-7-S4-S9.

33 Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1207–1215. SIAM, 2018. doi:10.1137/1.9781611975031.78.

34 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.

# What Else Can Voronoi Diagrams Do for Diameter in Planar Graphs?

**Amir Abboud**
Weizmann Institute of Science, Rehovot, Israel

**Shay Mozes**
Reichman University, Herzliya, Israel

**Oren Weimann**
University of Haifa, Haifa, Israel

## Abstract

The Voronoi diagrams technique, introduced by Cabello [SODA'17] to compute the diameter of planar graphs in subquadratic time, has revolutionized the field of distance computations in planar graphs. We present novel applications of this technique in static, fault-tolerant, and partially-dynamic undirected unweighted planar graphs, as well as some new limitations.

- In the static case, we give $n^{3+o(1)}/D^2$ and $\tilde{O}(n \cdot D^2)$ time algorithms for computing the diameter of a planar graph $G$ with diameter $D$. These are faster than the state of the art $\tilde{O}(n^{5/3})$ [SODA'18] when $D < n^{1/3}$ or $D > n^{2/3}$.
- In the fault-tolerant setting, we give an $n^{7/3+o(1)}$ time algorithm for computing the diameter of $G \setminus \{e\}$ for every edge $e$ in $G$ (the replacement diameter problem). This should be compared with the naive $\tilde{O}(n^{8/3})$ time algorithm that runs the static algorithm for every edge.
- In the incremental setting, where we wish to maintain the diameter while adding edges, we present an algorithm with total running time $n^{7/3+o(1)}$. This should be compared with the naive $\tilde{O}(n^{8/3})$ time algorithm that runs the static algorithm after every update.
- We give a lower bound (conditioned on the SETH) ruling out an amortized $O(n^{1-\varepsilon})$ update time for maintaining the diameter in *weighted* planar graph. The lower bound holds even for incremental or decremental updates.

Our upper bounds are obtained by novel uses and manipulations of Voronoi diagrams. These include maintaining the Voronoi diagram when edges of the graph are deleted, allowing the sites of the Voronoi diagram to lie on a BFS tree level (rather than on boundaries of $r$-division), and a new reduction from incremental diameter to incremental *distance oracles* that could be of interest beyond planar graphs. Our lower bound is the first lower bound for a dynamic planar graph problem that is conditioned on the SETH.

**2012 ACM Subject Classification** Theory of computation → Shortest paths; Theory of computation → Dynamic graph algorithms

**Keywords and phrases** Planar graphs, diameter, dynamic graphs, fault tolerance

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.4

**Related Version** *Full Version*: `https://arxiv.org/abs/2305.02946`

**Funding** *Amir Abboud*: Supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.
*Shay Mozes*: Israel Science Foundation grant 810/21.
*Oren Weimann*: Israel Science Foundation grant 810/21.

## 1 Introduction

The DIAMETER problem asks to compute the largest distance in the graph. It is one of the most basic and extensively studied problems in the graph algorithms literature, and moreover, it is prominent in Fine-grained Complexity where it has driven the development


of innovative hardness reductions [1,4,5,9,11,17,29,36,67]. Assuming the strong exponential time hypothesis (SETH), there is also no truly subquadratic algorithm for DIAMETER [5,67] in undirected, unweighted graphs with treewidth $\Omega(\log n)$. For graphs of bounded treewidth, the diameter can be computed in near-linear time [5] (see also [41,50] for algorithms with time bounds that depend on $D$). Near-linear time algorithms were developed for many other restricted graph families, see e.g. [8,14,31–34,40,43,49,66].

One of the outstanding questions that has remained open despite a decade of major developments in algorithms and conditional lower bounds for graph problems is whether DIAMETER can be solved in near-linear time in *planar graphs*. Until 2017, only logarithmic improvements over the natural $O(n^2)$ bound (of computing all-pairs shortest-path, APSP) had been known [23,72]. The consensus was that truly subquadratic time is impossible and the focus of the community was on proving a hardness result, e.g. under SETH. But then, in a celebrated paper, Cabello [22] gave a subquadratic $\tilde{O}(n^{11/6})$ time algorithm, that was later improved to the current-best $\tilde{O}(n^{5/3})$ bound [45].

The breakthrough in Cabello's work [22] is his novel use of *Voronoi Diagrams* (VDs) in planar graph algorithms. This new machinery has revolutionized the field of distance computation problems in planar graphs and has lead to several breakthroughs [26,28,35,47,63] including a surprising and almost-optimal *distance oracle* - a problem that had hitherto seen many gradual improvements using different techniques both in the exact [10,21,26,30,35,39, 42,47,56,63–65,73] and the approximate [24,48,54,55,58,69,74] settings. Consequently, the main meta question occupying the minds of researchers in planar graph algorithms is: *what else can Voronoi diagrams do for us?*

## 1.1 Dynamic Planar Diameter

It is natural to expect VDs to produce breakthroughs in the domain of *dynamic* planar graphs. Dynamic data structures that support updates and queries to a graph have remarkable applications in theory (as a subroutine in static algorithms) and practice (for changing inputs). Many ingenious algorithms for basic problems in dynamic planar graphs have been developed in the last few decades, including connectivity, distances, and cuts [6,18,19,25, 28,37,42,51–53,55,56,59,62,68,69], but large (polynomial) gaps remain compared to the lower bounds [3]. Only few of these works [27,28] use VDs and only in a limited way (they recompute the VD from scratch after every update). It is clear that major advancements await if one is able to maintain the VD machinery *dynamically* in a meaningful way. In this paper, we investigate this possibility by focusing on the DIAMETER problem.

The state-of-the-art algorithm recomputes the diameter from scratch after every update in time $\tilde{O}(n^{5/3})$. This is not surprising since the only useful technique against DIAMETER (in static graphs) is based on VDs, and we do not know how to make VDs dynamic.

The first question that comes to mind is: Suppose, optimistically, we could make VDs as dynamic as possible; *what time bound would we hope to get?* Clearly, we cannot get $O(n^{2/3-\varepsilon})$ time per update until we break the $\tilde{O}(n^{5/3})$ bound for static graphs. Moreover, a conditional $n^{2/3-o(1)}$ lower bound (under the APSP or Online Matrix Vector Conjectures) follows from the reductions of Abboud and Dahlgaard [3]. So perhaps dynamic VDs would lead to a matching $O(n^{2/3})$ upper bound? Our first result rules out this possibility with an $n^{1-o(1)}$ lower bound under SETH.

▶ **Theorem 1** (Lower Bound on Dynamic Diameter)**.** *If the diameter of a dynamic undirected planar graph on $n$ nodes can be maintained with $O(n^{1-\varepsilon})$ amortized time per weight-change, then SETH is false. This holds even if the dynamic algorithm is allowed to preprocess the initial graph in poly$(n)$ time, and even in the partially-dynamic setting where weights only increase or only decrease.*

Notably, this is the first lower bound for a dynamic planar graph problem that is based on the SETH (as opposed to other conjectures) and only the second example of such a result if we consider *static* planar graph problems as well [2, 46].

**Towards Dynamic Voronoi Diagrams.** A large gap of $n^{2/3}$ remains despite our lower bound and it is likely that it can be closed if we can indeed make VDs dynamic.[1]

In this paper, we take a small (but arguably the first) step towards this goal: we give an efficient algorithm for updating the VD after the deletion of one edge in the graph, much faster than recomputing it from scratch. (We refer to Section 5 for an overview and all the details.) This small step already has interesting applications. While it applies for general (weighted) planar graphs, the applications we have found only gain an advantage in unweighted planar graphs.

A concrete application is a faster algorithm for the REPLACEMENT DIAMETER: given a graph $G$ return the diameter of $G \setminus \{e\}$, the graph obtained by removing the edge $e$, for all edges $e$. The trivial algorithm for this problem makes $O(n)$ calls to a static diameter algorithm, one for each edge, and achieves $\tilde{O}(n^{8/3})$ running time. We improve this upper bound by an $n^{1/3}$ factor to $n^{7/3+o(1)}$ by utilizing our efficient updates to VDs, along with other tricks that are also based on VDs (but not in a dynamic way).

▶ **Theorem 2** (Replacement Diameter). *Given an unweighted undirected planar graph $G = (V, E)$, there is an $n^{7/3+o(1)}$ time algorithm that for every edge $e \in E$ outputs the diameter of $G^e = (V, E \setminus \{e\})$.*

An additional new result is a faster algorithm for DIAMETER in the *incremental* setting where we start from an empty graph and need to maintain the diameter while $O(n)$ edges are being added (without violating the planarity). The trivial algorithm recomputes the diameter after every update in a total of $\tilde{O}(n^{8/3})$ time, and we improve it to $n^{7/3+o(1)}$.

▶ **Theorem 3** (Incremental Diameter). *There is an algorithm that maintains the diameter of an unweighted undirected planar graph undergoing edge insertions in a total of $n^{7/3+o(1)}$ time.*

This result is based on an elegant reduction from incremental DIAMETER to incremental *distance oracles* that could be of interest beyond planar graphs. Its analysis relies on recent works on the *bipartite independent set* queries introduced by Beame et al. [13].

## 1.2 Static Planar Diameter

Back to DIAMETER in static graphs, what else can we hope to get from VDs? Of course, the biggest open question is whether the $n^{5/3}$ bound can be improved to $n^{1+o(1)}$, or whether one can prove a super-linear lower bound. Toward this question, we would like to understand the hard/easy cases, and a natural parameter to consider is $D$ – the diameter itself.

One of the main algorithmic contributions of this paper, that is crucial to the aforementioned upper bounds, is an algorithm beating $n^{5/3}$ when $D$ is large (in the range $[n^{2/3+\varepsilon}, n]$). Notably, it implies that anyone seeking a tight conditional lower bound cannot use constructions with very large diameter.

[1] It is tempting to think that Theorem 6 implies a dynamic diameter algorithm with update time $\tilde{O}(n^{1.6})$; Use an $r$-division and maintain for each piece the DDG and bisectors. Upon an update of an edge in a piece $P$, recompute the DDG of $P$ (using MSSP) and the bisectors of $P$ (using Theorem 6). For each vertex in the graph, recompute all additive weights using FR-Dijkstra, and compute the furthest vertex in each piece using Theorem 6. The caveat is that this approach does not handle properly the case where both endpoints of the diameter path belong to the same piece (not necessarily $P$). The reason is that the VD mechanism only handles paths that visit at least one boundary node.

▶ **Theorem 4** (Static Large Diameter). *The diameter can be computed in $n^{3+o(1)}/D^2$ time on an unweighted undirected planar graph with diameter $D$.*

Our new algorithm applies VDs in a novel way, where the VD sites lie on a BFS tree level, as opposed to lying on the boundary of pieces in an $r$-divisions.

While our result is the first to address the large $D$ case, the other extreme of small $D$ has already been studied. Eppstein [41] gave the first near-linear time algorithm for constant $D$, with an exponential dependence on $D$. This dependency was later improved as a byproduct of new $(1+\varepsilon)$-approximation algorithms for DIAMETER [15, 24, 41, 70]. The state of the art is $\tilde{O}(n \cdot D^5)$ using the $(1+\varepsilon)$-approximation $\tilde{O}(n \cdot (1/\varepsilon)^5)$-time algorithm of Chan and Skrepetos [24] with $\varepsilon = 1/D$. The final result of this paper is an improved bound of $\tilde{O}(nD^2)$ which increases the range in which the $n^{5/3}$ bound can be beaten from $D < n^{2/15-\varepsilon}$ to $D < n^{1/3-\varepsilon}$.

▶ **Theorem 5** (Static Small Diameter). *The diameter can be computed in $\tilde{O}(n \cdot D^2)$ time on an unweighted undirected planar graph with diameter $D$.*

Our algorithm exploits VDs in a more natural way than that of Chan and Skrepetos [24], if our goal is solve the small $D$ case exactly (recall that their focus is on approximations). It remains an interesting open question whether the $\tilde{O}(n \cdot (1/\varepsilon)^5)$ time approximation algorithm can be improved. This is related to another challenge of computing *approximate VDs* faster than exact, which we do not address in this paper.

## 2 Preliminaries

A recursive decomposition tree $\mathcal{T}$ of a planar graph $G$ is the tree obtained (in linear time) by recursively separating $G$ with a separator of size $\sqrt{|G|}$. $\mathcal{T}$ is a binary tree whose nodes correspond to subgraphs of $G$ (called *pieces*), with the root being all of $G$ and the leaves being pieces of constant size. We identify each piece $P$ with the node representing it in $\mathcal{T}$ (we can thus abuse notation and write $P \in \mathcal{T}$), and with its boundary $\partial P$ (i.e. vertices that belong to some separator along the recursive decomposition used to obtain $P$). An important property for us (see e.g. [47, Lemma 3.1]) is that the sum of $|P| \cdot |\partial P'|$ over all pairs of siblings $P, P'$ in $\mathcal{T}$ is $\tilde{O}(n^{1.5})$.

An *$r$-division* [44] of a planar graph $G$ is a decomposition of $G$ into $\Theta(n/r)$ pieces, each of them with $O(r)$ vertices and $O(\sqrt{r})$ boundary vertices (vertices shared with other pieces). It is possible to compute an $r$-division in $O(n)$ time [57] with the useful property that the boundary vertices of each piece lie on a constant number of faces of the piece (called *holes*).

The *dense distance graph* (DDG) of a piece $P$ is the complete graph over the boundary vertices of $P$. The length of edge $uv$ in the DDG of $P$ equals to the $u$-to-$v$ distance inside $P$. Note that the DDG of $P$ is non-planar. The DDG of an $r$-division is the union of DDGs of all pieces of the $r$-division. Thus, the total number of vertices in the DDG is $O(n/\sqrt{r})$, and the total number of edges is $O(n)$. The DDG of an $r$-division can be computed in $\tilde{O}(n)$ time using the MSSP algorithm [56]. Fakcharoenphol and Rao [42] described an $\tilde{O}(n/\sqrt{r})$ time implementation of Dijkstra's algorithm (nicknamed FR-Dijkstra) on the DDG.

The difficult case for computing the diameter is when the furthest pair of vertices lie in different pieces. Consider some source vertex $s$ outside of some piece $P$. For every boundary vertex $u$ of $P$, let $d(u)$ denote the $s$-to-$u$ distance in $G$. The *additively weighted Voronoi diagram* of $P$ with respect to $d(\cdot)$ is a partition of the vertices of $P$ into pairwise disjoint sets (Voronoi cells), each associated with a unique boundary vertex (site) $u$. The vertices in the cell $\mathrm{Vor}(u)$ are all the vertices $v$ of $P$ such that $u$ is the last boundary vertex of $P$ on the shortest $s$-to-$v$ path. In other words, every site $u$ of $P$ has *additive weight* $d(u)$, the

*additive distance* from a site $u$ to a vertex $v$ of $P$ is defined as $d(u)$ plus the length of the shortest $u$-to-$v$ path inside $P$, and the cell $\mathrm{Vor}(u)$ contains all vertices $v$ of $P$ that are closer (w.r.t. additive distances) to $u$ than to any other site in $S$. The *boundary* $\partial\mathrm{Vor}(u)$ of a cell $\mathrm{Vor}(u)$ consists of all edges of $P$ that have exactly one endpoint in $\mathrm{Vor}(u)$. For example, in a Voronoi diagram of just two sites $u$ and $v$, the boundary of the cell $\mathrm{Vor}(u)$ is a $uv$-cut and is therefore a cycle in the dual graph. This cycle is called the *uv-bisector*. The complexity $|\partial\mathrm{Vor}(u)|$ of a Voronoi cell $\mathrm{Vor}(u)$ is the number of faces of $P$ that contain vertices of $\mathrm{Vor}(u)$ and of at least two more Voronoi cells. For every source $s$, computing the furthest vertex from $s$ in $P$ thus boils down to computing, for each site $u$, the furthest vertex (w.r.t. additive distance) from $u$ in $\mathrm{Vor}(u)$, and then returning the maximum value among all sites $u$.

▶ **Theorem 6** ([45]). *Let $P$ be an edge-weighted planar graph with $r$ vertices. Let $S$ be a set of $b$ sites that lie on the boundaries of $\tilde{O}(1)$ faces[2] of $P$. The uv-bisectors of all pairs $u, v \in S$ and all possible additive weights $d(u), d(v)$ can be computed and represented in $\tilde{O}(rb^2)$ time and space. Then, given any additive weights $d(\cdot)$ to $S$, a representation of the Voronoi diagram w.r.t these weights can be constructed in $\tilde{O}(|S|)$ time. With this representation, for any site $u \in S$ we can query the maximum distance from $u$ to a vertex in $\mathrm{Vor}(u)$ in $\tilde{O}(|\partial\mathrm{Vor}(u)|)$ time.*

# 3 Static Diameter

## 3.1 An $n^{3+o(1)}/D^2$ Algorithm

In this subsection we prove Theorem 4, stating that the diameter can be computed in $n^{3+o(1)}/D^2$ time on an unweighted undirected planar graph with diameter $D$. We first present a randomized $\tilde{O}(n^4/D^3)$ time algorithm, and then show how to improve it to $n^{3+o(1)}/D^2$. We then show how to derandomize both algorithms. We begin with two simple observations about the BFS levels when the diameter is $\geq D$.

▶ **Observation 7.** *Let $s$ be any node in a graph of diameter $\geq D$. Then at least one out of the $D/2$ middle levels of the BFS tree rooted at $s$ has size $O(n/D)$.*

▶ **Observation 8.** *Let $s$ be any node in $G$ and let $L_i$ be the set of nodes at level $i$ in the BFS tree rooted at $s$. Let $G_i$ be the subgraph of $G$ that is induced by $\bigcup_{j\geq i} L_j$. Then for each connected component $C$ of $G_i$ the nodes in $L_i \cap C$ lie on a single face.*

**Proof.** To see that the vertices of $L_i \cap C$ all lie on the same face in $G_i$, consider the embedding of the component $C$ of $G_i$ inherited from the embedding of $G$. Viewing $C$ as a graph obtained from $G$ by deleting edges and vertices, one can start from any vertex of $L_i$ and follow a curve in the plane that only goes through deleted edges and vertices until reaching the root $s$ of the BFS tree. Hence all vertices of $L_i$ lie on a single face of $C$, and hence also of $G_i$. ◀

**A randomized algorithm.** We first compute in $O(n)$ time a 2-approximation (lower bound) $\tilde{D}$ of $D$ by computing a BFS tree and choosing $\tilde{D}$ to be the furthest root-to-leaf distance. Then, we repeat the following procedure $\theta(n \log n/\tilde{D})$ times, and return the largest distance found:

[2] Theorem 1.1 in [45] is phrased for a constant number of faced (called holes). However, as pointed in footnote 8 in [45], the dependency of the running time on the number of holes is polynomial, so the theorem applies also to the case of a polylogarithmic number of holes.

1. Randomly sample a source $s$, compute its BFS tree. Let $D'$ be the depth of this tree. Note that $D \geq D' \geq D/2$. Let $S = L_i$ be the set of nodes at level $i$ satisfying both $D'/4 < i < 3D'/4$ and $|S| = O(n/D') = O(n/D)$. By Observation 7, such a set exists. Let $G_i$ be the subgraph of $G$ induced by $\bigcup_{j \geq i} L_j$.
2. Compute $d(v, b)$ for all $v \in G$ and all $b \in S$.
3. For each connected component $C$ of $G_i$:
   a. Compute all bisectors in $C$ of sites $C \cap S$ (that lie on a single face by Observation 8).
   b. For each node $v$ in $G \setminus G_i$, compute the VD of $C$ w.r.t the additive weights $d(v, b)$, and compute the distance from $v$ to its furthest vertex in every Voronoi cell of the VD.

**Running time.** The first step takes $O(n)$ time by computing and traversing the BFS tree of $s$. The second step takes $O(n^2/D)$ time by doing a BFS from each vertex of $S$ in $O(n)$ time. The most expensive step is 3a. By Theorem 6, all bisectors of a connected component $C$ can be computed in $\tilde{O}(|C| \cdot |C \cap S|^2)$ time. Over all connected components, this sums up to $\tilde{O}(n \cdot (n/D)^2)$ (since the $C$'s are disjoint and sum up to $n$, and the $C \cap S$ are disjoint and sum up to $O(n/D)$). Finally, in step 3b, for each vertex $v$, computing $v$'s VD and furthest vertex in every cell takes $\tilde{O}(|C \cap S|)$ time by Theorem 6. Over all connected components, this sums up to $\tilde{O}(n/D)$, and thus over all vertices $v$ to $\tilde{O}(n^2/D)$. The total running time of the entire procedure is thus $\tilde{O}(n \cdot (n/D)^2)$, and since we repeat the procedure $\tilde{O}(n/D)$ times we get $\tilde{O}(n^4/D^3)$.

**Correctness.** It remains to prove that the distance we return is indeed the diameter with high probability. Let $x, y$ be the two endpoints of the diameter (i.e. $D = d(x, y)$). Then, the probability that a random source $s$ satisfies $d(s, x) \leq D'/4$ and $d(s, y) \geq 3D'/4$ is at least $D'/4n$ (because this happens if $s$ is one of the first $D'/4$ nodes on the path from $x$ to $y$). Therefore, this happens with high probability for at least one of the sources $s$ that we choose. For this $s$, we will have that $x \in G \setminus G_i$ while $y \in G_i$ (it is impossible that $y \in G \setminus G_i$ because then an $x$-to-$y$ path through $s$ would be shorter than $D$), and then the largest distance that we find is guaranteed to be $d(x, y)$.

**Derandomization.** Observe that to derandomize the algorithm, it suffices to replace the sampling of sources with a (deterministic) selection of a set of sources $\mathcal{S}$ of size $O(n/D)$ such that a diameter endpoint $x$ is at distance $\leq D'/4$ from at least one source $s \in \mathcal{S}$.

To construct $\mathcal{S}$, pick an arbitrary source $s$ and compute it's BFS tree $T$ of depth $D' \leq D$. Find a level $L_i$ that has only $O(n/D') = O(n/D)$ nodes and $0.4D' \leq i \leq 0.5D'$. Similarly, find a level $L_j$ that has only $O(n/D)$ nodes and $0.8D' \leq j \leq 0.9D'$. The set of sources is then $\mathcal{S} = \{s\} \cup L_i \cup L_j$. It is easy to verify that every vertex $v$ in the graph has an ancestor or a descendant in $T$ that belongs to $\mathcal{S}$ and is at distance at most $D'/4 \leq D/4$ from $v$.

**A faster algorithm.** Next, we improve the running time to $n^{3+o(1)}/D^2$. Again, we will start with a randomized algorithm and then derandomize. Let $B_\rho(v)$ denote the ball with radius $\rho$ around vertex $v$. Recall that our goal is to sample w.h.p. a vertex $s$ in $B_{\tilde{D}/4}(x)$ (without knowing $x$), where $x$ is a diameter endpoint.

Let $\rho = \tilde{D}/4$. In order to sample a vertex $s$ in $B_\rho(x)$ w.h.p., it suffices to randomly sample a set of $\tilde{O}(n/|B_\rho(x)|)$ vertices (rather than sampling $\tilde{O}(n/\rho)$ vertices as in the approach above). Then, for each sampled vertex $s$, we can find a level $L_i$ in the BFS tree of $s$ with $\rho < i \leq 2\rho$ s.t. $|L_i| < |B_{2\rho}(s)|/\rho$ (rather than $n/\rho$ as in the approach above). Then, executing the approach above (i.e., executing steps 2–3 of the $\tilde{O}(n^4/D^3)$ algorithm above) for a specific

$s$ would take time $\tilde{O}(n(|B_{2\rho}(s)|/\rho)^2)$ to compute all bisectors, $\tilde{O}(n|B_{2\rho}(s)|/\rho)$ to compute all additive weights, and $\tilde{O}(|B_{2\rho}(s)|^2/\rho)$ to construct the Voronoi diagrams for all vertices above level $i$. We see that if $|B_\rho(x)|$ is large then we gain because we have to sample fewer vertices, and if $|B_{2\rho}(s)|$ is small then we gain because the amount of work for each sampled vertex decreases.

For this approach to work, we need to (1) estimate $|B_\rho(x)|$, and (2) relate $|B_\rho(x)|$ and $|B_{2\rho}(s)|$. To address (1), we simply estimate $|B_\rho(x)|$ by enumerating all powers of two $2^k$ for $0 \le k \le \log n$. To address (2), note that $|B_\rho(x)| < |B_{2\rho}(s)| < |B_{3\rho}(x)|$, and that there must exist a $j \in \{1, 2, \ldots, \sqrt{\log_3 n}\}$ s.t. $|B_{3\rho 3^{-j}}(x)|/|B_{\rho 3^{-j}}(x)| < 3^{\sqrt{\log_3 n}} = n^{o(1)}$ (if not, $|B_\rho(x)| > n$, a contradiction).

The algorithm is therefore: For each $1 \le j \le \sqrt{\log_3 n}$, let $\rho_j = 3^{-j}\rho$. For each $0 \le k < \log n$ we sample $(n \log n)/2^k$ vertices $s$ (reflecting our assumption that $B_{\rho_j}(x) \le 2^k$). For each sampled vertex $s$, if $|B_{2\rho_j}(s)| > 2^k 3^{\sqrt{\log_3 n}}$, then, since $|B_\rho(x)| < |B_{2\rho}(s)| < |B_{3\rho}(x)|$, it must be that $s \notin B_{\rho_j}(x)$ or $|B_{\rho_j}(x)| > 2^k$ or $|B_{\rho_{j-1}}(x)|/|B_{\rho_j}(x)| > 3^{\sqrt{\log_3 n}}$ (the disjunction is not exclusive). Hence, in this case we discard $s$ and move on to the next sampled vertex. Otherwise, $|B_{2\rho_j}(s)| \le 2^k 3^{\sqrt{\log_3 n}}$, and we can find a level $L_i$ with $\rho_j < i < 2\rho_j$ in the BFS tree rooted at $s$ s.t. $|L_i| < 2^k 3^{\sqrt{\log_3 n}}/\rho_j$, and continue as in steps 2–3 from the previous algorithm. The overall running time is

$$\sum_{j=0}^{\sqrt{\log_3 n}} \sum_{k=0}^{\log n} \tilde{O}\left(\frac{n}{2^k}\left(n(2^k 3^{\sqrt{\log_3 n}}/\rho_j)^2 + n2^k 3^{\sqrt{\log_3 n}}/\rho_j + (2^k 3^{\sqrt{\log_3 n}})^2/\rho_j\right)\right) = n^{3+o(1)}/D^2.$$

To argue correctness, note that for $j$ such that $|B_{\rho_{j-1}}(x)|/|B_{\rho_j}(x)| \le 3^{\sqrt{\log_3 n}}$ and $k$ such that $2^{k-1} \le |B_{\rho_j}(x)| \le 2^k$, sampling $(n \log n)/2^k$ vertices will yield with high probability a vertex $s \in B_{\rho_j}(x)$, and this $s$ will not be discarded. This $s$ satisfies $d(s,x) \le \rho_j$ and $d(s,y) \ge 2\rho_j$, so the largest distance found for this $s$ is guaranteed to be $d(x,y)$ by the same argument as in the correctness of the slower algorithm.

**Derandomization.** We use sparse neighborhood covers of Busch, Lafortune and Tirthapura [20] to derandomize the algorithm. A $\rho$-neighborhood cover $Z$ of a graph $G$ is a set of connected subgraphs called clusters, such that the union of all clusters is the vertex set of $G$ and such that for each node $v \in G$, there is some cluster $C \in Z$ that contains $B_\rho(v)$. The radius of a cover $Z$ is the maximum radius of a cluster in $Z$. The degree of a cover $Z$ is the maximum number of clusters that a node in $G$ is a part of. Busch et al. gave a deterministic $O(n \log n)$-time algorithm for computing, for any $\rho > 0$ and any connected planar graph, a $\rho$-neighborhood cover of any connected planar graph with radius $16\rho$ and degree 18. See also [60] for an $O(n)$ time algorithm.

To adjust the arguments we redefine $\rho_j = \rho 33^{-j}$ for $j = 1, \ldots, \sqrt{\log_{33}(n)}$, and use the fact that for some $j$, $|B_{\rho_{j-1}}|/|B_{\rho_j}| < 33^{\sqrt{\log_{33} n}}$. To avoid sampling in our algorithm, for each choice of $j, k$, we compute a $\rho_j$-neighborhood cover $Z$. We pick an arbitrary vertex $s$ from each cluster $C$ of $Z$ such that $|C| > 2^k$. Since the degree of $Z$ is 18, the number vertices $s$ we choose is at most $18n/2^k$.

If $2^k < |B_{\rho_j}(x)| > 2^{k+1}$ then the cluster $C$ containing $B_{\rho_j}(x)$ will have $|C| > 2^k$ vertices, and we will choose a vertex $s \in C$. Since the radius of $Z$ is $16\rho_j$, $d(s,x) \le 16\rho_j$. If $|B_{17\rho_j}(s)| > 2^{k+1} 33^{\sqrt{\log_{33} n}}$, we discard $s$. Since $B_{17\rho_j}(s)$ is contained in $B_{33\rho_j}(x) = B_{\rho_{j-1}(x)}$, we are guaranteed that some $s$ will not be discarded. For such $s$ we find a level $L_i$ with $16\rho_j < i < 17\rho_j$ in the BFS tree rooted at $s$ s.t. $|L_i| < 2^{k+1} 33^{\sqrt{\log_{33} n}}/\rho_j$. The level of $x$ in

the BFS tree is at most $16\rho_j$, and since $\rho_j < \rho < D/4$, the vertex $y$ such that $d(x,y) = D$ is at level greater than $i$ in the BFS tree. Hence, executing lines 2–3 of the procedure for the algorithm in section 5 will report the distance $D$. The running time analysis is identical to that of the randomized version since we made sure that the number of vertices we choose in the derandomiztion is at most some fixed constant times the number of sampled vertices in the randomized algorithm.

## 3.2 An $\tilde{O}(n \cdot D^2)$ Algorithm

In this subsection we prove Theorem 5, stating that the diameter can be computed in $\tilde{O}(n \cdot D^2)$ time on an unweighted planar graph with diameter $D$. We begin with some preliminaries on a recursive decomposition using shortest path separators.

**Preliminaries.** A *shortest path separator* of a planar graph $G$ is an undirected cycle $C(G)$ consisting of a shortest $s$-to-$u$ path, a shortest $s$-to-$v$ path, and a single edge $uv$, such that both the interior and exterior of the cycle consist of at most 2/3 of the total number of the faces of $G$. Such a separator can be found in $O(n)$ time [61]. By recursively separating $G$ with shortest path separators (halting the recursion when we reach subgraphs of size $\leq D$), we obtain the *decomposition tree* $\mathcal{T}$. The root of $\mathcal{T}$ corresponds to the entire graph $G$. A node corresponding to subgraph $P$ (we interchangeably refer to it as node $P$) has two children, whose subgraphs correspond to the interior and exterior of the separator $C(P)$.

Observe that for every node $P \in \mathcal{T}$ the size of the shortest path separator $C(P)$ is $O(D)$. This is because $C(P)$ consists of two shortest paths, each of length at most $D$. Moreover, the boundary of $P$ (vertices of $P$ that have incident edges to vertices not in $P$) is included in the union of all $C(P')$ where $P'$ is an ancestor of $P$, and is therefore of size $O(D \log n)$ and lies on $O(\log n)$ faces of $P$. We compute the DDGs of every node (subgraph) $P \in \mathcal{T}$ (i.e. copmute a data structure that can report in $\tilde{O}(1)$ time the distances in the graph $P$ between and pair of boundary vertices of $P$) using $O(\log n)$ executions of MSSP on $P$. This takes total $\tilde{O}(n)$ time over the entire $\mathcal{T}$. Now, given any vertex $v$ in the subgraph $P$, we can compute the distances in $G$ from $v$ to all boundary vertices of $P$ in $\tilde{O}(D)$ time using FR-Dijkstra. Namely, we initialize the $\tilde{O}(D)$ boundary vertices of $P$ to their distances from $v$ in the graph $P$ (via MSSP queries), and we run FR-Dijkstra on the union of the DDG of $P$ and the DDGs of all $P'$ where $P'$ is a sibling of some ancestor of $P$.

**The algorithm.** For every non-leaf node $P \in \mathcal{T}$, we compute the furthest pair of vertices $u, v \in P$ where $u$ is internal to $C(P)$ and $v$ is external to $C(P)$. Observe that distances must be taken in the entire graph $G$ since the shortest $u$-to-$v$ path may venture out of $P$. To this end, we precompute all bisectors of every graph $P \in \mathcal{T}$, with the sites being the $\tilde{O}(D)$ boundary vertices of $P$. Using Theorem 6, this takes $\tilde{O}(|P| \cdot D^2)$ time (where $|P|$ denotes the size of the subgraph $P$), so over all $\mathcal{T}$ this takes $\tilde{O}(n \cdot D^2)$ time. (Observe that here we have used Theorem 6 with the sites lying on $O(\log n)$ faces. As far as we know, in all prior uses of Theorem 6 the sites lie on $O(1)$ faces). Then, for every vertex $v \in P$, we compute the distances in $G$ from $v$ to all boundary vertices of $P$ using FR-Dijkstra in $\tilde{O}(D)$ time as explained above. We then use these distances as additive weights and apply Theorem 6 to find the furthest vertex from $v$ in $P$. This also takes $\tilde{O}(D)$ time, so overall $\tilde{O}(n \cdot D)$.

We handle the leaf nodes $P \in \mathcal{T}$ explicitly (recall that $|P| \leq D$). For each leaf node $P$ we compute the all-pairs shortest-paths (APSP) in $G$ between any two vertices $u, v \in P$. This is done by running Dijkstra's standard algorithm from every $v \in P$ on the graph $P$ where the boundary vertices of $P$ are initialized to their distances from $v$ in $G$ (that we have already computed as $v$'s additive weights). This takes $\tilde{O}(D)$ time per $v$, so $\tilde{O}(D^2)$ time per $P$, and $\tilde{O}(D^2 \cdot n/D) = \tilde{O}(nD)$ over all leaves $P$.

## 4 A Lower Bound on Dynamic Diameter

In this section we prove Theorem 1. Namely, we give a conditional lower bound ruling out an amortized $O(n^{1-\varepsilon})$ update time for maintaining the diameter of *weighted* planar graphs that undergo a sequence of edge-weight updates.

The proof is inspired by [3], however, there are quite a few changes since the reduction in [3] is from APSP (not SETH), to dynamic distance oracles (not dynamic diameter), and rules out $O(n^{0.5-\varepsilon})$ update time (not $O(n^{1-\varepsilon})$). Our reduction is from the following problem, which is simply a recasting of the Orthogonal Vectors problem in the language of graphs.

▶ **Definition 9** (Graph OV). *Given an undirected tripartite graph $G$ with parts $A, C, B$ where $|A| = |B| = n$ and the middle level has size $|C| = O(\log n)$, where all edges are in $A \times C$ and $C \times B$ decide if there exists a pair $a_i \in A, b_j \in B$ such that $d_G(a_i, b_j) > 2$.*

It is known that solving this Graph OV problem in $O(n^{2-\varepsilon})$ time refutes SETH [67, 71]. Moreover, in the unbalanced version where $|A| = n^\alpha$ and $|B| = n^\beta$ for arbitrary constants $\alpha, \beta > 0$ we know that an $O(n^{\alpha+\beta-\varepsilon})$ time algorithm refutes SETH.

**The structure of the reduction.** Given an instance $G$ of the Graph OV problem, we construct a dynamic planar graph $H$. The graph $H$ is composed of two grids, a left grid and a right grid, each of dimension $|C| = O(\log n)$ by $|A| = n$. The columns of both grids are indexed by the nodes of $A$, such that the top node of the $i^{th}$ column in the left (resp. right) grid is called $a_i$ (resp. $a'_i$). The rows of the grids correspond to the nodes in $C$ such that the rightmost (resp. leftmost) node in the $k^{th}$ row of the left (resp. right) grid is called $c_k$ (resp. $c'_k$). In both grids, all horizontal edges have weight $2|C|$. In the left grid, the vertical edges in column $i$ have weight $2i$ and in the right grid the vertical edges of column $i$ have weight $2(n-i)$. In the left grid, for every $i$ and $k$, if the edge $(a_i, c_k)$ exists in $G$, then we add a diagonal edge $e_k$ from vertex $(k-1, i)$ to vertex $(k, i+1)$ whose weight is $2i + 2|C| - 1$. We call such $e_k$ a *shortcut* edge (as it is shorter by 1 compared to the alternative path composed of a vertical edge followed by a horizontal edge). The two grids are connected by $|C|$ edges: for each $k$ we have an edge from $c_k$ to $c'_k$ of weight $2n|C| - 2nk$. These $|C|$ edges are the only edges in $H$ whose weights will change throughout the reduction - all others will remain fixed. We add a single node $x$ that is connected to all nodes in the top row of the left grid and all nodes of the top row in the right grid. We set the weight of every edge $(a_i, x)$ to be $i \cdot 4|C|$ and the weight of every edge $(x, a'_j)$ to be $(n-j) \cdot 4|C|$.

**The dynamic updates.** After constructing the initial graph $H$ as above, for every $j = 1, \ldots, n$ we obtain a graph $H_j$ by applying the following updates to $H$: for every $k = 1, \ldots, |C|$ if the edge $(c_k, b_j)$ exists in $G$ then decrease by 1 the weight of the edge $(c_k, c'_k)$ in $H$ (we refer to such edge $(c_k, c'_k)$ as a *decreased* edge). The following main lemma shows that the diameter of $H_j$ reveals whether or not there exists an $a_i \in A$ such that $d_G(a_i, b_j) > 2$. Note, crucially, that we can generate all graphs $H_1, \ldots, H_n$ in sequence using only $O(n \log n)$ updates since $H_i$ differ from $H_{i-1}$ by only $O(\log n)$ edge weights. Under SETH, we cannot maintain the diameter throughout this sequence in $O(n^{2-\varepsilon})$ time. Therefore, each update cannot be done in $O(n^{1-\varepsilon})$ amortized time, thus proving Theorem 1 for the fully-dynamic case. To get a proof for the incremental case where edge weights only decrease we can do the following (the decremental case is symmetric). Redefine the weight of the $O(\log n)$ edges so that they only decrease during the sequence: add $2(n-i)$ to their weight in $H_i$ so that their weight is the largest in $H_1$ and smallest in $H_n$. Then, the sequence of graphs can be generated by only

$O(n \log n)$ decrease-weight updates. The diameter of $H_i$ increases by exactly $2(n-i)$ so the same analysis goes through. For simplicity, we continue the proof in this section with the construction in the fully-dynamic case.

▶ **Lemma 10.** *For any $j$, the diameter of $H_j$ is larger than $4n|C|-2$ iff there exists $a_i \in A$ such that $d_G(a_i, b_j) > 2$.*

In the remainder of this section we prove the above lemma. First observe that by our choice of edge-weights the diameter of $H_j$ correspond to some shortest $a_i$-to-$a'_\ell$ path. The following claim shows that in fact it is an $a_i$-to-$a'_i$ path.

▷ Claim 11. For all $i \neq \ell$, $d_{H_j}(a_i, a'_\ell) < 4n|C|-2$.

Proof. If $\ell > i$, then the path $a_i - x - a'_\ell$ consisting of two edges costs $(n-(\ell-i)) \cdot 4|C| < 4n|C|-2$. Otherwise $\ell < i$, then the $a_i$-to-$a'_\ell$ path that only uses horizontal edges costs $2|C|(n-i+\ell) + 2|C| = 2n < 4n|C|-2$. ◁

The following claim concludes the proof.

▷ Claim 12. For any $i$, $d_{H_j}(a_i, a'_i) > 4n|C|-2$ iff $d_G(a_i, b_j) > 2$.

Proof. Observe that the path $a_i - x - a'_i$ consisting of two edges costs $4n|C|$. There may however be a shorter $a_i$-to-$a'_i$ path that passes through the grids. By our choice of edge weights (similarly to [3]) such shortest path must start with an $a_i$-to-$c_k$ prefix (for some $k \leq |C|$) in the left grid, then use the $(c_k, c'_k)$ edge, then in the right grid do $i$ horizontal steps followed by $k$ vertical steps. Moreover, the $a_i$-to-$c_k$ prefix starts with $k-1$ vertical steps, then uses a shortcut edge $e_k$ if it exists (otherwise it does a horizontal step followed by a vertical step), and then it does $n-i-1$ horizontal steps until reaching $c_k$.

Suppose first that there were no shortcut edges and no decreased edges at all. The length of such $a_i$-to-$a'_i$ path would then be

$$d_H(a_i, a'_i) = k \cdot 2i + (n-i) \cdot 2|C| + (2n|C| - 2nk) + i \cdot 2|C| + k \cdot 2(n-i) = 4n|C|.$$

Note that this length ($4n|C|$) is the same independent of $k$ and of $i$. Hence, the only way an $a_i$-to-$a'_i$ path can be shorter than $4n|C|$ is by using shortcut edges and decreased edges. However, it can use at most one shortcut edge $e_k$ and one decreased edge $(c_k, c'_k)$. So its length is $4n|C|-2$ iff there exists a $k$ such that the shortcut $e_k$ exists (i.e., $(a_i, c_k) \in E(G)$) and the edge $(c_k, c'_k)$ is decreased (i.e., $(c_k, b_j) \in E(G)$), and this is iff $d_G(a_i, b_j) \leq 2$. ◁

▶ Remark 13. By subdividing all edges, the above reduction implies that $O(n^{1/2-\varepsilon})$ update time is impossible for maintaining the diameter of *unweighted* planar graphs.

## 5 Decremental Voronoi diagrams and Replacement Diameter

**Overview: A Step Toward Dynamic Voronoi Diagrams.** The usefulness of Voronoi diagrams for diameter and distance reporting in static planar graphs make it natural to ask whether one can efficiently maintain some useful representation of Voronoi diagrams in the dynamic setting. This seems challenging because a change in a single edge or in a single additive weight can cause the entire Voronoi diagram to completely change. For example, decreasing the weight of a single edge in the Voronoi cell $\mathrm{Vor}(b)$ of some site $b$ may cause an expansion of $\mathrm{Vor}(b)$ on the expense of every other Voronoi cell, even cells that were not neighbors of $\mathrm{Vor}(b)$ before the change. The same is true for decreasing the additive weight of $b$. Indeed, the few attempts to use Voronoi diagrams in dynamic planar graphs that we are aware of [27, 28], all recompute the Voronoi diagrams from scratch upon every update.

We make a small step towards dynamic Voronoi diagrams by developing a mechanism for updating Voronoi diagrams in the decremental setting. In our opinion, this is the most novel technical contribution of our work. The deletion of an edge in some part of the graph only causes an increase in the additive weights of certain sites. When the additive weight of site $b$ increases, its Voronoi cell shrinks. Namely, some vertices that were in $\mathrm{Vor}(b)$ before the increase will belong to Voronoi cells of other sites after the increase. The crucial observation is that the only relevant sites in this process are $b$ and the sites of the neighboring cells to $\mathrm{Vor}(b)$ in the original Voronoi diagram. The time for the resulting update procedure is therefore proportional to the cell-degree of $b$, rather than to the total number of sites in the Voronoi diagram. Unfortunately, the cell-degree of $b$ may, in general, be as large as the number of sites. Nonetheless, this procedure turns out to be useful for the replacement diameter problem, where we can bound the number of times each site is affected by some edge deletion.

**Representing Voronoi diagrams.** Let $P$ be a planar graph with real edge-lengths. Let $S$ be the set of vertices (*sites*) that lie on $\tilde{O}(1)$ faces, called holes. Recall that every site $s \in S$ has an associated additive weight $d(s)$.

Consider the Voronoi diagram of $P$ with sites $S$ and additive weights $d(s)$. Let $P^*$ be the planar dual of $P$. Let $\mathrm{VD}_0$ be the subgraph of $P^*$ consisting of the duals of edges $(u, v)$ of $P$ such that $u$ and $v$ are in different Voronoi cells. Let VD be the graph obtained from $\mathrm{VD}_0$ after eliminating all degree-2 vertices by repeatedly contracting any one of their incident edges. The vertices of VD are called *Voronoi vertices* and the edges of VD are called *Voronoi edges*. Observe that every Voronoi edge corresponds to a consecutive segment of some bisector between two sites. Note that VD may be disconnected, i.e., a planar map, and that the boundaries of faces of this planar map may be disconnected. Each face of VD corresponds to a cell $\mathrm{Vor}(s)$ of some site $s \in S$. Hence there are at most $|S|$ faces in VD. It is shown in [45] that the total number of edges, vertices, and faces of VD is $O(|S|)$. In what follows, when we say we compute a Voronoi diagram VD, we mean we use the algorithm in Theorem 6, which computes a representation of the planar map VD defined above. Each Voronoi edge of VD corresponds to a segment of a bisector.

## 5.1 Maintaining Voronoi diagrams while additive weights increase

Consider an increase in the additive weight of a set $B \subseteq S$ of sites. Such an increase can only change the shortest path (w.r.t. additive weights) of vertices $v$ in the Voronoi cells of sites in $B$. Either the shortest path to such $v$ remains the same but its length increases by the change in the additive weight of the site, or $v$ becomes part of a Voronoi cell of a different site. In the latter case, since each shortest path is entirely contained in a single Voronoi cell, planarity dictates that the new site must be a neighbor of a site in $B$. We define the set $N(B)$ of neighbors of the sites in $B$ as the set of sites that are either in $B$ or sites whose Voronoi cells are adjacent to the Voronoi cells of sites in $B$. Note that $|N(B)| = O(\sum_{b \in B} \text{cell-degree}(b))$. It follows from the discussion above that the only sites whose Voronoi cells might change as a result of such an increase are those in $N(B)$.

To compute the new Voronoi diagram we first compute the Voronoi diagram of $P$ with just the sites $N(B)$. By Theorem 6, this takes $O(\sum_{b \in B} \text{cell-degree}(b))$ time. Let $\mathrm{VD}'$ denote this Voronoi diagram, and let VD denote the Voronoi diagram of $P$ before the change. We stress that the additive weights of $\mathrm{VD}'$ are the ones after the increase, and the additive weights of VD are the ones before the increase. To obtain the Voronoi diagram of $P$ after the change, we "glue" together parts of $\mathrm{VD}'$ and VD as follows. See Figure 1 for an illustration.

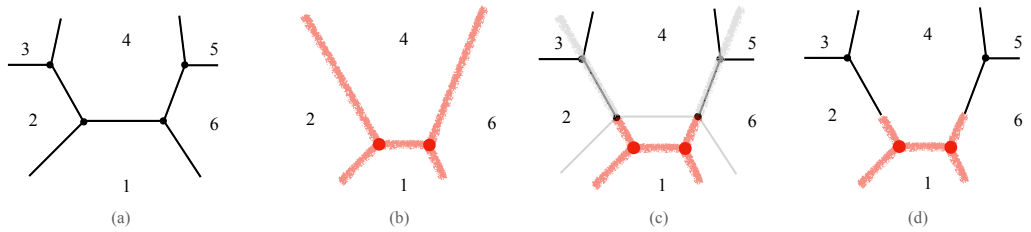



**Figure 1** An illustration of the process of computing the Voronoi diagram of a piece with 6 sites when the additive weight of site 1 is increased. (a) VD, the Voronoi diagram of all 6 sites before the weight increase. (b) VD$'$, the Voronoi diagram of just the increased site (1) and its neighbors (2, 4, 6), after the increase. (c) VD and VD$'$ superimposed, with the edges deleted from VD, and from VD$'$ in grey. Observe that all segments of bisectors between cells of the neighbors (2,4,6) that appear in VD also appear in VD$'$. (d) The glued Voronoi diagram.

Recall that VD is a (possibly disconnected) planar map whose edges correspond to segments of bisectors of pairs of sites of VD. The endpoints of these segments are Voronoi vertices of VD. We start by deleting from VD all the Voronoi edges corresponding to bisectors involving at least one site of $B$. For every Voronoi vertex $v$ incident to a Voronoi cell of a site in $B$, if all the Voronoi edges incident to $v$ were deleted, then we delete $v$ as well. Let $\mathcal{E}$ denote the set of Voronoi edges $e$ of VD such that $e$ is incident to some Voronoi vertex $v$, $e$ was not deleted, but the preceding or following Voronoi edge of $e$ in the cyclic order of edges around $v$ was deleted. Every Voronoi edge $e \in \mathcal{E}$ corresponds to a segment $\beta$ of a bisector between two sites $s_1, s_2 \in N(B) \setminus B$. Since the additive weights of these sites are unchanged, the segment $\beta$ must be represented by a Voronoi edge $e'$ of VD$'$. Note that $\beta$ may be a sub-segment of the bisector segment $\beta'$ corresponding to $e'$. Also note that it is easy to identify $e'$ with $e$ during the computation of VD$'$ with no asymptotic time overhead.[3] For each Voronoi edge $e \in \mathcal{E}$ (of VD), we split its corresponding Voronoi edge $e'$ (of VD$'$) into two edges $e'_1, e'_2$ by breaking $\beta'$ into two sub-segments at $v$. Suppose $e'_2$ is the one whose corresponding bisector segment contains $\beta$. Note that if $v$ is an endpoint of $e'$ (i.e., if $v$ is a Voronoi vertex of VD$'$ as well), then $e'_1$ is a trivial empty segment of the $s_1$-$s_2$ bisector. We delete $e'_2$ from VD$'$, and merge the Voronoi edges $e$ of VD and $e'_1$ of VD$'$ into a single Voronoi edge whose corresponding segment is the concatenation of the segment $\beta$ of $e$ and the segment of $e'_1$.

Doing so for the edges $e \in \mathcal{E}$ effectively "glues" the relevant portion of VD$'$ into VD, replacing the portion of VD that we had deleted. The algorithm of [45] for constructing Voronoi diagrams from precomputed bisectors performs similar stitching and glueing operations, and the data structures used to represent Voronoi diagrams and bisectors support all the necessary operations in $\tilde{O}(1)$ time per operation. Hence, the time complexity of this entire procedure is proportional (up to logarithmic factors) to the number of Voronoi vertices of the Voronoi cells of the sites in $B$, which is $O(\sum_{b \in B} \text{cell-degree}(b))$.

[3] This can be done, for example, by augmenting the binary search tree representation of segments of bisectors used in the construction algorithm (cf. [45]) with a boolean flag marking edges in $\mathcal{E}$. Then we can go over all Voronoi edges of VD$'$ and list for each one the corresponding marked edge $e \in \mathcal{E}$, if such an edge exists in the segment of the bisector corresponding to that Voronoi edge.

## 5.2 Replacement Diameter

We now describe how to use the new algorithm for maintaining Voronoi diagrams under additive weight decreases to get a faster algorithm for replacement diameter. The algorithm starts by computing a complete recursive decomposition tree $\mathcal{T}$ of the graph $G$. For every node (piece) in $\mathcal{T}$ (corresponding to a subgraph of $G$) we compute all its bisectors. This takes $\tilde{O}(n^2)$ time over all $\mathcal{T}$ using Theorem 6. Then, for every vertex $s \in V$ we compute the BFS tree $T_s$ of $s$ in $G$ and compute the *fault-tolerant single source distance oracle* of Baswana et. al. [12] for $s$ in $G$. This oracle is constructed in $\tilde{O}(n)$ time from $G$, and can report in $\tilde{O}(1)$ time the $s$-to-$t$ distance in the graph $G^e = (V, E \setminus \{e\})$ for any $s, t \in V$ and any $e \in E$. Overall, this also takes $\tilde{O}(n^2)$ time. For each piece $P \in \mathcal{T}$, for each boundary vertex $b \in \partial P$ we create the *induced tree* $T_b^P$ from $T_b$ by marking all vertices of $P$ and all their lowest common ancestors, and contracting any edge whose endpoints are not marked. The resulting $T_b^P$ has size $O(|P|)$. For each edge $e$ of $G$ that was contracted in the process we store the edge of $T_b^P$ into which $e$ was contracted. Since the total number of boundary nodes and piece sizes over all pieces of $\mathcal{T}$ is $\tilde{O}(n)$, the total time to construct all these induced trees is $\tilde{O}(n^2)$. For each piece $P \in \mathcal{T}$, let $P'$ be the sibling of $P$ in $\mathcal{T}$. Let $b_1, b_2, \ldots$ be the vertices of $\partial P'$ in some arbitrary order. For each vertex $s \in P$ we compute the additivley weighted Voronoi diagram of $s$ w.r.t $P'$ with sites $\{b_i\}$ and additive weights $d(s, b_i)$. We also store for $s$ a binary search tree (BST) over $b_1, b_2, \ldots$, where the node $i$ in the tree stores the distance from $s$ to the furthest vertex in $\mathrm{Vor}(b_i)$. This takes total $\tilde{O}(n\sqrt{n})$ time over all $P \in \mathcal{T}$ and all $s \in P$. For each piece $P$ with vertices $v_1, v_2, \ldots$ in arbitrary order, we store a BST over $\{v_i\}$, where node $i$ stores the furthest vertex from $v_i$ in $P'$. This vertex can be found in $\tilde{O}(1)$ time for each $v_i$ by querying the maximum distance stored in the BST of $v_i$.

For every edge $e \in E$, we need to compute the furthest pair of vertices in the graph $G^e = (V, E \setminus \{e\})$. For an edge $e \in E$ and two vertices $u, v \in G$, we say that the pair $u, v$ is *affected* in $G^e$ if $e$ lies on the root-to-$v$ path in $T_u$. The main idea is to use the fact that a specific pair $u, v$ is affected in at most $D$ (rather than $n$) graphs $G^e$ (since the shortest $u$-to-$v$ path in $G$ has at most $D$ edges).

For every affected pair $(u, v)$ there is some pair of sibling pieces $(P, P')$ s.t. $u \in P$ and $v \in P'$. Our strategy is to go over pairs of sibling pieces $(P, P')$ in $\mathcal{T}$, and handle all affected pairs for each $(P, P')$ together as follows. Assume w.l.o.g. that $e \notin P'$. For each $b \in \partial P'$, we enumerate in $T_b^P$ all the decendant vertices of the edge of $T_b^P$ into which $e$ was contracted (this may be an empty set if $e \notin T_b$). This way we identify all the affected pairs of the form $(u, b)$, where $u \in P$ and $b \in \partial P'$. We query the Baswana et al. oracle for the $u$-to-$b$ distance in $G^e$ for each such affected pair. For each $u \in P$, let $B$ be the set of boundary vertices $b$ such that $(u, b)$ is an affected pair. For each vertex $u \in P$ with $|B| \geq 1$, we update the Voronoi diagram of $u$ w.r.t. $P'$ using the procedure Decremental-VD, which is described in subsection 5.1. This procedure updates the VD (and the furthest vertex from each site) w.r.t the new additive weights in time $\sum_{b \in B} \text{cell-degree}(b)$ where cell-degree is the number of Voronoi cells that are adjacent[4] to the cell $\mathrm{Vor}(b)$ in the original VD (i.e. before the deletion of $e$). Using the updated VD, we update the node corresponding to every $b \in B$ in the BST of $u$ with the new furthest vertex in $\mathrm{Vor}(b)$. Let $d$ be the maximum distance stored in the entire BST of $u$. We update the node corresponding to $u$ in the BST of $P$ with the value $d$. After handling all $u \in P$ with $|B| \geq 1$ in this way, the maximum value stored in the entire BST of $P$ is the maximum distance in $G^e$ between any pair of vertices $(u, v)$ with $u \in P$ and $v \in P'$. Taking the maximum over all pairs of siblings $(P, P') \in \mathcal{T}$ gives the diameter of $G^e$.

[4] Two cells are adjacent if there exists an edge $e$ of $G$ with one endpoint in each cell.

The total running time for computing the furthest pair for the siblings $(P, P')$ is analyzed as follows. The bottleneck is the time to update the VDs. Every time a pair $u, b$ (where $u \in P$ and $b \in \partial P'$) is affected we spend $\tilde{O}(\text{cell-degree}(b))$ time updating the VD of $u$. Since each pair is affected by the deletion of at most $D$ edges, the total time invested in updating VDs for $(P, P')$ is bounded by $\sum_{u \in P, b \in \partial P'} D \cdot \text{cell-degree}(b) = |P|D \sum_b \text{cell-degree}(b)$, which is $\tilde{O}(|P|D \cdot |\partial P'|)$, since the sum of cell-degrees of the cells in a VD is order of the number of sites of the VD. Summing over all pairs of sibling pieces we get that the total time is $\sum_{(P,P') \in \mathcal{T}} \tilde{O}(|P|D \cdot |\partial P'|) = \tilde{O}(n^{1.5}D)$. Hence, including the preprocessing, the total time for the entire replacement diameter algorithm is $\tilde{O}(n^2 + n^{1.5}D)$.

We note that when $D \geq n^{5/6}$, it is better to naively use the static $n^{3+o(1)}/D^2$-time algorithm from section 3.1 for each edge failure. Hence, replacement diameter can be solved in $\min(n^{3+o(1)}/D^2, \tilde{O}(n^2 + n^{1.5}D)) = n^{7/3+o(1)}$ time.

## 6 Incremental Diameter

In this section we prove Theorem 3. Namely, we present a general reduction showing how to solve the diameter problem efficiently in *incremental* graphs given two components: (1) a distance oracle for incremental graphs, and (2) a diameter algorithm for static graphs that is relatively fast when the diameter is large. Plugging in the incremental distance oracle of Das et al. [37] and the static algorithm of Section 3.1 we obtain an algorithm with total time $n^{7/3+o(1)}$ which improves over the naive bound of $\tilde{O}(n^{8/3})$. The new algorithm of this section comes closer to the $n^{2-o(1)}$ lower bound of Section 4 for weighted graphs (the best lower bound for unweighted graphs is $n^{1.5-o(1)}$).

The rest of this section is dedicated to proving this theorem. We begin by presenting the general reduction (that does not assume planarity nor unweighted edges) and then explain how it can be combined with existing algorithms for planar graphs to obtain the theorem.

**A reduction from diameter to $s, t$-shortest path.** In an incremental graph, the diameter decreases with time, starting from some $D \leq n$ (otherwise the graph is not connected and it is easy to check this efficiently) and ending at some $D \geq 1$. The idea for the reduction is simple: we would like to recompute the diameter only when it decreases, and not after each of the $n$ updates. While it is true that the diameter could decrease $\Omega(n)$ times, from $n$ to 1, the point is that re-computation is efficient when the diameter is large (due to the $n^{3+o(1)}/D^2$ algorithm of Section 3.1) and then only $O(D)$ of the re-computations will happen when the diameter is smaller than $D$.

Our incremental algorithm works as follows:

- **Step 1 - sample a new diameter pair:** Let $P = \{(s,t) \mid d(s,t) = \Delta(G)\}$ be the set of pairs that realize the current diameter $\Delta(G)$. Sample a pair $(s', t')$ from $P$ uniformly at random (or from some distribution in which every pair is sampled with probability at most $O(1/|P|)$).
- **Step 2 - monitor the distance of the sampled pair:** Using an incremental distance oracle, monitor the distance between $s'$ and $t'$ throughout the sequence of edge insertions. Do nothing (except querying the oracle) as long as $d(s', t')$ does not decrease; in which case it is still the correct diameter of the graph and can be output whenever there is a query. If a new edge causes $d(s', t')$ to decrease, go back to Step 1.

Each of the two steps involves one of the two ingredients in our reduction. Step 2 utilizes an incremental distance oracle, while Step 1 uses a static diameter algorithm *that can also sample a diameter pair*. At the end of this section we give a general reduction from the

latter approximate sampling problem to the problem of finding the largest distance from each node in the graph (i.e. computing all eccentricities). Alternatively, one could notice that the diameter algorithms we will employ in Step 1 (and many other natural diameter algorithms) can be modified to also sample a diameter pair uniformly at random.

**Running time.** Let us first bound the number of times we go to Step 1, which is the most costly step since it involves a static diameter computation. Step 2 is actually very cheap since we only perform one update and one query to an incremental distance oracle.

▷ Claim 14. For any (non adaptive) sequence of edge insertions that does not decrease the diameter of the graph, the expected number of times our algorithm samples a diameter pair (i.e. goes to Step 1) is $O(\log n)$.

Proof. Let us first analyze the idealistic case in which we manage to sample truly uniformly in Step 1, and then point out that the same analysis essentially goes through when we sample almost uniformly.

Each new edge $e$ decreases the distance for a subset of pairs $X_e \subseteq P$. Since the special pair $(s', t')$ is completely unknown to the adversary who is choosing the sequence of edge insertions, the probability that $e$ causes the algorithm to go to Step 1 is exactly $|X_e|/|P|$ and in that case the new set of "diameter pairs" becomes $P \setminus X_e$. Therefore, the expected number of times we sample can be upper bounded by: $f(|P|) \leq \max_{0 \leq x \leq |P|} x/|P| + f(|P| - x) = O(\log |P|)$.

If the sampling in Step 1 is only approximately uniform, but still satisfies that a pair is chosen with probability at most $O(1/|P|)$ then the same analysis above goes through, up to an additional $O(1)$ factor. ◁

Let $T^{Diam}(n, D)$ denote the running time of a static diameter algorithm that samples a diameter pair as in Step 1, when the diameter of the graph is $D$. Over all the $O(n)$ edge insertions, the total expected running time of Step 1 is therefore at most $\sum_{D=1}^{n} \log n \cdot T^{Diam}(n, D)$.

To obtain our claimed upper bound of $n^{7/3+o(1)}$ we will use two diameter algorithms inside this reduction: the $T^{Diam}(n, D) = n^{3+o(1)}/D^2$ algorithm from Section 3.1 (for large $D$) and the $T^{Diam}(n, D) = \tilde{O}(n^{5/3})$ algorithm [45] (for small $D$). (By the reduction in Section 6.1, these algorithms can also sample an approximately uniform pair as required by Step 1). The total expected time becomes:

$$\sum_{D=1}^{n} \log n \cdot T^{Diam}(n, D) = \tilde{O}\left(\sum_{D=1}^{n^{2/3}} n^{5/3} + \sum_{D=n^{2/3}}^{n} n^{3+o(1)}/D^2\right) = n^{7/3+o(1)},$$

because $\sum_{D=n^{2/3}}^{n} n^{3+o(1)}/D^2 \leq \sum_{i=\log_2 n^{2/3}}^{\log_2 n} 2^{i+1} \cdot n^{3+o(1)}/(2^i)^2 \leq \frac{n^{3+o(1)}}{n^{2/3}} \cdot 2 \log n$. The additional time of Step 2 is at most $n \cdot \sqrt{n}$ using the incremental distance oracle of Das et al. [37] that has $O(\sqrt{n})$ time per update and query.

## 6.1 Sampling a Diameter Pair

In this section we show the final piece of the incremental diameter algorithm. Namely, a way to adapt the aforementioned static diameter algorithms so that they sample a diameter pair approximately uniformly.

A first attempt, that does not quite work, is to add a random "perturbation" $p_e \in (0, \varepsilon)$ to the weight of each edge $e$, where $\varepsilon < 1/D$, and then argue that the (probably unique) pair realizing the diameter in the new graph is a uniformly random pair in $P = \{(s, t) \mid d(s, t) =$

$\Delta(G)\}$. Note that the perturbations increase the distance between all pairs by $< 1$ and therefore non-diameter-pairs cannot become diameter pairs. One issue, however, is that pairs with many paths of length $\Delta(G)$ between them are more likely to be chosen than pairs with few such paths. A second attempt that resolves this issue is to add a perturbation to the nodes (e.g. by appending a private leaf to each node with a random weight on the new edge). This idea is closer to the actual solution but it still has an issue of correlations: a node that participates in many pairs might be sampled less frequently than a node that participates in few pairs. Therefore, we must take this difference into account when assigning the weights.

Making these ideas go through is a bit complicated. Fortunately, there is an elegant reduction from our setting to the *bipartite independet set* query model introduced by Beame et al. [13] and then use existing results on this model [7, 16, 38] in a black-box way.

▶ **Theorem 15.** *There is an algorithm that samples a pair in* $P = \{(s,t) \mid d(s,t) = \Delta(G)\}$ *where each pair is sampled with probability at most* $O(1/|P|)$ *and runs in time* $(\min(\tilde{O}(n^{5/3}), n^{3+o(1)}/D^2))$ *on unweighted planar graphs of diameter* $D$*.*

The main lemma towards proving the theorem is the following.

▶ **Lemma 16.** *By making* $\log^{O(1)} n$ *calls to an algorithm that returns all eccentricities we can sample a pair in* $P = \{(s,t) \mid d(s,t) = \Delta(G)\}$ *where each pair is sampled with probability at most* $O(1/|P|)$

**Proof.** Consider an implicit graph $H$ in which there is an edge between two nodes $s, t$ iff they are a diameter pair in $G$ (i.e., $(s,t) \in P$). Our goal is to sample an edge from $H$ approximately uniformly. This can be achieved [7, 16, 38] by making a polylogarithmic number of queries to an oracle that, given two subsets $L, R \subseteq V(H)$, decides whether there is any edge in $L \times R \cap E(H)$. This is called a bipartite independent set oracle in the literature, following Beame et al. [13]. Thus, all we have to do is show that such a query can be supported in the time of a call to an algorithm that computes all eccentricities in the graph.

First, we precompute the diameter $\Delta(G)$ of $G$. Then, given a query $L, R \subseteq V$ we construct a graph $G'$ from $G$ as follows. For each node $v \in R$ we add a new "leaf" node $l_v$ and connect it with an edge (of weight 1) to $v$. Next, we compute the eccentricity of all nodes in $G'$. Finally, the answer to the query is yes if and only if there is a $u \in L$ such that the eccentricity of $u$ in $G'$ is $\Delta(G) + 1$; this can be checked in $O(n)$ time.

The correctness of the answer follows from the observation that the eccentricity of any node $u$ in $G'$ is $\Delta(G) + 1$ if and only if there is a node $v$ in $G$ such that (1) $d_G(u,v) = \Delta(G)$ and (2) a new leaf node $l_v$ was appended to $v$. This implies that (1) $(u,v) \in P$ is a diameter pair in $G$, meaning that $(u,v) \in E(H)$, and that (2) $v \in L$. Since we only check for $u \in R$ our answer that $L \times R \cap E(H)$ is non-empty is correct. ◀

To conclude the proof of Theorem 15 we simply point out that both of the relevant diameter algorithms already compute the eccentricity of all nodes.

## References


1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *ACM Trans. Algorithms*, 17(4):30:1–30:40, 2021. doi:10.1145/3469834.

2 Amir Abboud, Vincent Cohen-Addad, and Philip N. Klein. New hardness results for planar graph problems in P and an algorithm for sparsest cut. In *52nd STOC*, pages 996–1009, 2020.

3 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *57th FOCS*, pages 477–486, 2016.

4 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443, 2014.

5 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA*, pages 377–391, 2016.

6 Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *44th annual ACM symposium on Theory of computing*, pages 1199–1218, 2012.

7 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. *arXiv preprint arXiv:2207.02817*, 2022.

8 A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

9 Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In *46th ICALP*, volume 132, pages 13:1–13:14, 2019.

10 Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *4th ESA*, volume 1136, pages 514–528, 1996.

11 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Toward tight approximation bounds for graph diameter and eccentricities. *SIAM J. Comput.*, 50(4):1155–1199, 2021. doi:10.1137/18M1226737.

12 Surender Baswana, Utkarsh Lath, and Anuradha S. Mehta. Single source distance oracle for planar digraphs avoiding a failed node or link. In *23rd SODA*, pages 223–232, 2012.

13 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Transactions on Algorithms (TALG)*, 16(4):1–27, 2020.

14 Boaz Ben-Moshe, Binay K. Bhattacharya, Qiaosheng Shi, and Arie Tamir. Efficient algorithms for center problems in cactus networks. *Theor. Comput. Sci.*, 378(3):237–252, 2007.

15 P. Berman and S.P Kasiviswanathan. Faster approximation of distances in graphs. In *Proc. of the 10th International Workshop on Algorithms and Data Structures (WADS)*, pages 541–552, 2007.

16 Anup Bhattachrya, Arijit Bishnu, Arijit Ghosh, and Gopinath Mishra. Faster counting and sampling algorithms using colorful decision oracle. *Leibniz International Proceedings in Informatics (LIPIcs)*, 219, 2022.

17 Édouard Bonnet. 4 vs 7 sparse undirected unweighted diameter is SETH-hard at time $n^{4/3}$. *ACM Trans. Algorithms*, 18(2):11:1–11:14, 2022. doi:10.1145/3494540.

18 Glencora Borradaile, Seth Pettie, and Christian Wulff-Nilsen. Connectivity oracles for planar graphs. In *13th SWAT*, volume 7357, pages 316–327, 2012.

19 Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. In *51th FOCS*, pages 601–610, 2010.

20 Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Sparse covers for planar graphs and graphs that exclude a fixed minor. *Algorithmica*, 69(3):658–684, 2014. doi:10.1007/s00453-013-9757-4.

21 Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012. doi:10.1007/s00453-010-9459-0.

22 Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *28'th SODA*, pages 2143–2152, 2017.

23 Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Trans. Algorithms*, 8(4):34:1–34:17, 2012.

24 Timothy M. Chan and Dimitrios Skrepetos. Faster approximate diameter and distance oracles in planar graphs. *Algorithmica*, 81(8):3075–3098, 2019. doi:10.1007/s00453-019-00570-z.

25 Hsien-Chih Chang, Robert Krauthgamer, and Zihan Tan. Almost-linear $\epsilon$-emulators for planar graphs. In *54th STOC*, pages 1311–1324, 2022.

26 Panagiotis Charalampopoulos, Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Almost optimal distance oracles for planar graphs. In *51st STOC*, pages 138–151, 2019.

27 Panagiotis Charalampopoulos and Adam Karczmarz. Single-source shortest paths and strong connectivity in dynamic planar graphs. *J. Comput. Syst. Sci.*, 124:97–111, 2022. doi:10.1016/j.jcss.2021.09.008.

28 Panagiotis Charalampopoulos, Shay Mozes, and Benjamin Tebeka. Exact distance oracles for planar graphs with failing vertices. In *30th SODA*, pages 2110–2123, 2019.

29 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *25th SODA*, pages 1041–1052, 2014.

30 Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *32nd STOC*, pages 469–478, 2000.

31 Victor Chepoi and Feodor F. Dragan. A linear-time algorithm for finding a central vertex of a chordal graph. In *ESA*, pages 159–170, 1994.

32 Victor Chepoi, Feodor F. Dragan, Bertrand Estellon, Michel Habib, and Yann Vaxès. Diameters, centers, and approximating trees of delta-hyperbolicgeodesic spaces and graphs. In *SoCG*, pages 59–68, 2008.

33 Victor Chepoi, Feodor F. Dragan, and Yann Vaxès. Center and diameter problems in plane triangulations and quadrangulations. In *SODA*, pages 346–355, 2002.

34 K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, ii. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

35 Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *58th FOCS*, pages 962–973, 2017.

36 Mina Dalirrooyfard, Ray Li, and Virginia Vassilevska Williams. Hardness of approximate diameter: Now for undirected graphs. In *62nd FOCS*, pages 1021–1032, 2021. doi:10.1109/FOCS52979.2021.00102.

37 Debarati Das, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. A near-optimal offline algorithm for dynamic all-pairs shortest paths in planar digraphs. In *33rd SODA*, pages 3482–3495, 2022.

38 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colorful decision oracle. *SIAM J. Comput.*, 51(4):849–899, 2022. doi:10.1137/19m130604x.

39 Hristo Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *22nd WG*, volume 1197, pages 151–165, 1996.

40 Feodor F. Dragan and Falk Nicolai. Lexbfs-orderings of distance-hereditary graphs with application to the diametral pair problem. *Discrete Applied Mathematics*, 98(3):191–207, 2000.

41 D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In *6th SODA*, pages 632–640, 1995.

42 J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006.

43 Arthur M. Farley and Andrzej Proskurowski. Computation of the center and diameter of outerplanar graphs. *Discrete Applied Mathematics*, 2(3):185–191, 1980.

44 G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, 1987.

45 Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic $\tilde{O}(n^{5/3})$ time. *SIAM J. Comput.*, 50(2):509–554, 2021.

46 Pawel Gawrychowski, Shay Mozes, and Oren Weimann. Planar negative $k$-cycle. In *32nd SODA*, pages 2717–2724, 2021.

**47** Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *SODA*, 2018.

**48** Qian-Ping Gu and Gengchun Xu. Constant query time $(1+\varepsilon)$-approximate distance oracle for planar graphs. *Theor. Comput. Sci.*, 761:78–88, 2019.

**49** John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM J. Comput.*, 26(6):1612–1634, 1997.

**50** Thore Husfeldt. Computing graph distances parameterized by treewidth and diameter. In *IPEC*, pages 16:1–16:11, 2016.

**51** Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *49th STOC*, pages 1108–1121, 2017.

**52** Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *43rd STOC*, pages 313–322, 2011.

**53** Adam Karczmarz. Decrementai transitive closure and shortest paths for planar digraphs and beyond. In *29th SODA*, pages 73–92, 2018.

**54** Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *38th ICALP*, volume 6755, pages 135–146, 2011.

**55** Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *24th SODA*, pages 550–563, 2013.

**56** P. N. Klein. Multiple-source shortest paths in planar graphs. In *16th SODA*, pages 146–155, 2005.

**57** P. N. Klein, S. Mozes, and C. Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *45th STOC*, pages 505–514, 2013.

**58** Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *13th SODA*, pages 820–827, 2002.

**59** Jakub Lacki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. *Theory Comput. Syst.*, 61(4):1037–1053, 2017.

**60** Hung Le and Christian Wulff-Nilsen. Optimal approximate distance oracle for planar graphs. In *62nd FOCS*, pages 363–374, 2021. doi:10.1109/FOCS52979.2021.00044.

**61** R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36(2):177–189, 1979.

**62** J. Łącki and P. Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In *19th ESA*, pages 155–166, 2011.

**63** Yaowei Long and Seth Pettie. Planar distance oracles with better time-space tradeoffs. In *32nd SODA*, pages 2517–2536, 2021.

**64** Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd SODA*, pages 209–222, 2012.

**65** Yahav Nussbaum. Improved distance queries in planar graphs. In *12th WADS*, pages 642–653, 2011.

**66** Stephan Olariu. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34:121–128, 1990.

**67** Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *45th STOC*, pages 515–524, 2013.

**68** Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *1st ESA*, volume 726, pages 372–383, 1993.

**69** Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.

**70** O. Weimann and R. Yuster. Approximating the diameter of planar graphs in near linear time. *ACM Trans. Algorithms*, 12(1):12:1–12:13, 2016.

**71** Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In *31st ICALP*, pages 1227–1237, 2004.

**72** C. Wulff-Nilsen. Wiener index and diameter of a planar graph in subquadratic time. Technical report, 08-16, Department of Computer Science, University of Copenhagen, 2008. Available at `http://www.diku.dk/OLD/publikationer/tekniske.rapporter/rapporter/08-16.pdf`. Preliminary version in EurCG 2009.

**73** Christian Wulff-Nilsen. *Algorithms for planar graphs and graphs in metric spaces*. PhD thesis, University of Copenhagen, 2010.

**74** Christian Wulff-Nilsen. Approximate distance oracles for planar graphs with improved query time-space tradeoff. In *27th SODA*, pages 351–362, 2016.

# Smooth Distance Approximation

**Ahmed Abdelkader**
Google LLC, Mountain View, CA, USA

**David M. Mount**
Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

## Abstract

Traditional problems in computational geometry involve aspects that are both discrete and continuous. One such example is nearest-neighbor searching, where the input is discrete, but the result depends on distances, which vary continuously. In many real-world applications of geometric data structures, it is assumed that query results are continuous, free of jump discontinuities. This is at odds with many modern data structures in computational geometry, which employ approximations to achieve efficiency, but these approximations often suffer from discontinuities.

In this paper, we present a general method for transforming an approximate but discontinuous data structure into one that produces a smooth approximation, while matching the asymptotic space efficiencies of the original. We achieve this by adapting an approach called the partition-of-unity method, which smoothly blends multiple local approximations into a single smooth global approximation.

We illustrate the use of this technique in a specific application of approximating the distance to the boundary of a convex polytope in $\mathbb{R}^d$ from any point in its interior. We begin by developing a novel data structure that efficiently computes an absolute $\varepsilon$-approximation to this query in time $O(\log(1/\varepsilon))$ using $O(1/\varepsilon^{d/2})$ storage space. Then, we proceed to apply the proposed partition-of-unity blending to guarantee the smoothness of the approximate distance field, establishing optimal asymptotic bounds on the norms of its gradient and Hessian.

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

**Keywords and phrases** Approximation algorithms, convexity, continuity, partition of unity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.5

**Related Version** *Full Version*: https://arxiv.org/abs/2308.08791

**Acknowledgements** Conducted in part while the first author was a postdoctoral fellow at the University of Texas at Austin, and completed before he joined Google LLC.

## 1 Introduction

The field of computational geometry has largely focused on computational problems with discrete inputs and outputs. Discrete structures are often used to represent geometric objects that are naturally continuous. Examples include using triangulated meshes to represent smooth surfaces, Voronoi diagrams to represent distance maps, and various spatial partitions for answering ray-shooting queries. Due to the high computational complexities involved, researchers often turn to approximation algorithms. Unfortunately, in retrieval problems, efficient approximation is often achieved at the expense of continuity.

To make this more precise, consider the common example of distance functions. For a given set $S \subseteq \mathbb{R}^d$ (which may be discrete or continuous), a natural *distance map* over $\mathbb{R}^d$ arises as:

$$d_S : x \mapsto \inf_{p \in S} \|x - p\|,$$

where $\|\cdot\|$ denotes the Euclidean norm. In turn, the distance map gives rise to the following query problem. Given a query point $x \in \mathbb{R}^d$, the objective is to compute $d_S(x)$ efficiently from a data structure of low storage.

It is well known that answering the distance query can be reduced to computing the Voronoi diagram of $S$. Unfortunately, beyond special low-dimensional cases, the combinatorial complexity of the Voronoi diagram grows too fast for practical use. For this reason, much work has focused on data structures for approximate nearest neighbor (ANN) searching [7, 9, 21, 23, 24]. Given any $\varepsilon > 0$, an $\varepsilon$-ANN data structure returns a point that is within a factor of $1 + \varepsilon$ of the true closest distance.

While approximate nearest-neighbor searching is clearly related to approximating the distance map, there are fundamental differences between the two problems. The distance map induced by any set is clearly continuous (and indeed it is 1-Lipschitz continuous [12]). As two query points converge on a common location, their respective distances to $S$ must also converge. The same cannot be said for any of the existing approaches based on approximate nearest neighbor searching. The ANN distances reported for two query points can differ by an amount that is arbitrarily larger than the distance between the two query points. In Section 1.1, we will show that this is not merely an artifact of the design of these data structures; it is unavoidable.

Answering distance queries efficiently is key to many applications including motion planning [45], surface reconstruction [3, 26], physical modeling [36], and data analysis [10, 18]. Discontinuities can result in various sorts of aberrant behaviors. This is because queries are generated adaptively in a feedback loop, where answers to earlier queries are used to determine subsequent queries. Consider, for example, a navigation system that is trying to precisely dock two crafts moving in space. Discontinuities in the distance map can alter the behavior of the feedback process, resulting in jittering, oscillations, and even infinite looping (see examples in Section 1.1).

This motivates the main question considered in this paper: Does there exist a data structure that answers distance queries approximately so that the induced distance function is continuous? Ideally, the distance function should also be smooth, characterized by bounds on the norm of its gradient and Hessian. Note that this is quite different from approximate nearest-neighbor searching, where the objective is to find a point that approximates the closest distance. Here, the objective is approximate the distance itself.

Applications of distance queries include collision detection [14], penetration depth [47], robot navigation [31, 42], shape matching [2], and density estimation [32]. Often, the set $S$ arises as a discrete point set obtained by sampling an underlying surface. Implicit representations of surfaces [13], based on approximating the induced distance map, have recently witnessed significant developments based on deep neural networks [16, 22, 37], where the properties of learned distance fields are yet to be fully understood [30, 38].

In this paper we present a general approach for smooth approximation from traditional non-continuous data structures. This is achieved through a process called *blending*, where discrete local approximations are combined to form a smooth function. Our method is loosely based on the *partition-of-unity method* (see, e.g., Melenk and Babuška [34]). The approach involves constructing an open cover of the domain by overlapping patches, computing a local approximation within each patch, and then blending these approximations together by associating a smooth weighting function with each patch (see Section 2 for details).

Unfortunately, a direct adaptation of these methods does not yield an efficient solution. To the best of our knowledge, existing work on partition-of-unity methods for distance approximation have not considered the asymptotic efficiency of the resulting access structures. These works have typically involved blending over relatively simple spatial decompositions, such as grids [40] and balanced quadtrees [35]. The covering elements employed in the blending were naturally fat, that is, *isotropic*. These subdivisions are particularly suitable for blending,

but they lack the flexibility needed to achieve the highest levels of efficiency. Moreover, we are not aware of prior results on the asymptotic interplay between approximation and smoothness. (We refer the interested reader to recent works in the finite element literature on anisotropic [46] and high-dimensional [27] refinements.) In this paper, we adopt the partition-of-unity approach to perform smooth blending for distance maps while achieving asymptotic complexity bounds that match the best existing approximation algorithms. Our results will be presented in Section 1.2.

## 1.1 On Discontinuities and Witnesses

To better understand how discontinuities arise, it is useful to understand the general structure of most data structures for answering distance queries. Space is subdivided into regions, or *cells*. This is either done explicitly by defining the subdivision over the query range or implicitly by viewing the data structure abstractly as a decision tree and associating each leaf of the tree with the subset of query points that land in this leaf due to the search process. Queries are answered by determining the cell (or cells) that are relevant to the answer, and accessing distance information for each cell. When the query point moves from one cell to another, even infinitesimally, different distance information is accessed, and the computed distance may change discontinuously.

For example, consider four point sites $P = \{p_1, p_2, p_3, p_4\}$ in $\mathbb{R}^2$. Suppose that we construct an $\varepsilon$-ANN data structure based on a subdivision into rectangular cells (see Figure 1(a)). We assume that each cell stores a single site of $P$, called a *representative*, that serves as an $\varepsilon$-ANN for every query point lying in this cell, and assume further that the representatives have been chosen as shown in the figure, with $q_i$'s representative being $p_i$. Suppose that a gradient descent algorithm is run using this structure. Starting from an initial position (e.g., $q_i$), the descent takes a step towards the cell's representative ($p_i$). If the representatives and step sizes are chosen as in the figure, the descent could loop infinitely.

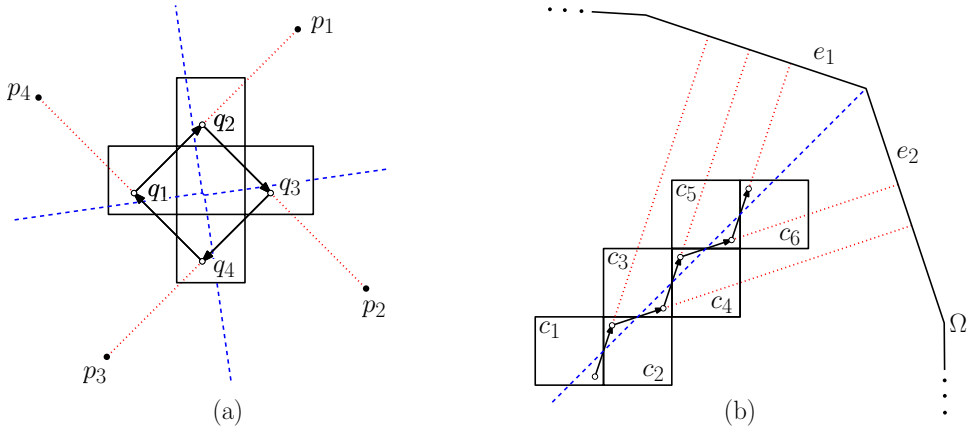



**Figure 1** Problems with witness-based distance approximation: (a) infinite loops and (b) jittering. (The dashed blue lines bound the Voronoi cells of the sites, and the dotted red lines indicate the direction to the closest site.)

In Figure 1(b), we consider another distance function computed with respect to the boundary of a convex object $\Omega$. Cells $c_1$, $c_3$, and $c_5$ are assigned edge $e_1$ as representative, and cells $c_2$, $c_4$, and $c_6$ are assigned $e_2$. If at each point we walk towards the closest edge to the cell's centroid, the path oscillates or "jitters" between the two contenders.

In both of these examples, we assume a standard model in which each cell stores a *witness* to an approximate nearest neighbor, and the distance function returns the distance from the query point to this witness. Let $\mathcal{Q}$ be a function that maps query points to witnesses (presumably based on the cell containing the query point), and let $\widetilde{d}_{\mathcal{Q}}$ denote the induced distance function $\widetilde{d}_{\mathcal{Q}}(x) = \|x - \mathcal{Q}(x)\|$. Such an approach is said to be *witness-based*. The following lemma shows that any witness-based method that fails to be exact cannot be both continuous and accurate with respect to relative errors.

▶ **Lemma 1.** *If a witness-based distance function $\widetilde{d}_{\mathcal{Q}}$ for a finite point set $P \subset \mathbb{R}^d$ is inexact at even one point, it cannot be both continuous and provide a finite bound on relative errors.*

**Proof.** Suppose towards a contradiction that $\widetilde{d}_{\mathcal{Q}}$ is continuous, guarantees a relative error of at most $c$ for some $c > 0$, but there exists a point $x \in \mathbb{R}^d$ such that $\widetilde{d}_{\mathcal{Q}}(x) > d_P(x)$. In particular, we may select an arbitrarily small $\delta > 0$ such that $\widetilde{d}_{\mathcal{Q}}(x) > d_P(x) + \delta$. Let $p \in P$ denote a nearest neighbor of $x$ and consider how the value $\widetilde{d}_{\mathcal{Q}}$ varies as we walk from $x$ to $p$ along the line segment $\overline{xp}$. More precisely, letting $u$ be a unit vector directed from $x$ to $p$, define $x(t) = x + t \cdot u$, and $\widetilde{d}_{\mathcal{Q}}(t) = \widetilde{d}_{\mathcal{Q}}(x(t))$. Except at a finite number of transition points where the witness changes, the derivative of $\widetilde{d}_{\mathcal{Q}}(t)$ with respect to $t$ cannot be smaller than $-1$. (A derivative of $-1$ occurs when we are walking straight towards the current witness, and otherwise it is strictly larger.) Since the function is continuous, its value does not change at transition points. It follows that as we travel a distance of $t \leq d_P(x)$ from $x$ to $p$, the value returned by $\widetilde{d}_{\mathcal{Q}}$ cannot decrease by an amount more than $t$. Setting $t = d_P(x) - \delta/c$, we conclude that

$$\widetilde{d}_{\mathcal{Q}}(x(t)) \;\geq\; \widetilde{d}_{\mathcal{Q}}(x) - t \;>\; (d_P(x) + \delta) - \left(d_P(x) - \frac{\delta}{c}\right) \;=\; \frac{(1+c)\delta}{c}.$$

But, $d_P(x(t)) = d_P(x) - t = \delta/c$, implying that the relative error is

$$\frac{\widetilde{d}_{\mathcal{Q}}(x(t)) - d_P(x(t))}{d_P(x(t))} \;=\; \frac{\widetilde{d}_{\mathcal{Q}}(x(t))}{d_P(x(t))} - 1 \;>\; \frac{(1+c)\delta}{c} \cdot \frac{c}{\delta} - 1 \;=\; (c+1) - 1 \;=\; c,$$

a contradiction. ◀

## 1.2 Main Result

For the sake of concreteness, we will illustrate our approach to producing smooth approximate distance functions in a specific application which is fairly simple, but still new. Let $\Omega$ denote a convex polytope in $\mathbb{R}^d$, and let $\mathrm{diam}(\Omega)$ denote its diameter and $\partial\Omega$ its boundary. We further assume that $\Omega$ is represented as the intersection of $n$ halfspaces. Given a point $x \in \Omega$, we define the *boundary distance function* $d_{\partial\Omega}(x)$ as the Euclidean distance to $x$'s closest point on $\partial\Omega$. To simplify notation, we will refer to this as $d_\Omega(x)$ (see Figure 2(a)). Our objective is to efficiently evaluate an $\varepsilon$-approximation $\widetilde{d}_\Omega$ for any given query $x \in \Omega$, while guaranteeing smoothness (i.e., continuity and norm bounds on the gradient and Hessian).

By convexity, if $x$ lies in $\Omega$'s interior, $\mathrm{int}(\Omega)$, its closest point on the boundary lies on one of $\Omega$'s facets, that is, its faces of dimension $d-1$. Thus, in the exact setting, the distance map is determined by the Voronoi diagram of $\Omega$'s facets. The skeleton of this Voronoi diagram is known as the *medial axis* or *medial diagram* of $\Omega$ [17, 19, 41]. While the combinatorial complexity of the medial axis is $O(n)$ in $\mathbb{R}^2$, it grows much faster in higher dimensions. It is not hard to show that medial axis corresponds to the lower-envelope of $n$ hyperplanes in $\mathbb{R}^{d+1}$, with a combinatorial complexity of $\Theta(n^{\lceil d/2 \rceil})$ in the worst case [33].

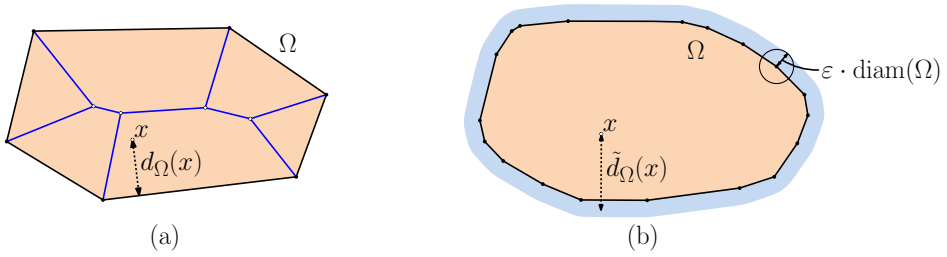



**Figure 2** (a) The medial axis of $\Omega$ and the boundary distance function $d_\Omega$ and (b) approximating the boundary distance in terms of absolute errors with parameter $\varepsilon > 0$.

The obvious discrete analog to our problem is approximate polytope membership, where the data structure merely indicates whether the query point lies inside or outside the polytope, up to a Hausdorff error of $\varepsilon \cdot \mathrm{diam}(\Omega)$ (see Figure 2(b)). In recent work, it was shown that this problem can be solved in query time $O(\log(1/\varepsilon))$ from a data structure using $O(1/\varepsilon^{(d-1)/2})$ of space [1, 5].

In this paper, we show how to apply the partition-of-unity method to evaluate an absolute $\varepsilon$-approximate boundary distance function $\widetilde{d}_\Omega$ for a convex polytope $\Omega$ in a manner that guarantees smoothness while nearly matching the query times achieved in approximate membership queries. Specifically, we require that $|\widetilde{d}_\Omega(x) - d_\Omega(x)| \leq \varepsilon \cdot \mathrm{diam}(\Omega)$, for all $x \in \mathrm{int}(\Omega)$. Throughout we treat $\varepsilon$ as an asymptotic quantity, and assume the dimension $d$ is a constant. Our main result is:

▶ **Theorem 2.** *Given a convex polytope $\Omega$ and an approximation parameter $\varepsilon > 0$, there exists a smooth function $\widetilde{d}_\Omega$ satisfying $d_\Omega(x) \leq \widetilde{d}_\Omega(x) \leq d_\Omega(x) + \varepsilon \cdot \mathrm{diam}(\Omega)$ for all $x \in \Omega$, which can be evaluated along with its gradient from a data structure with*

$$\textit{Query time} \;=\; O(\log(1/\varepsilon)) \qquad \textit{and} \qquad \textit{Storage} \;=\; O(1/\varepsilon^{d/2}).$$

*Further, the norms of the gradient and Hessian of $\widetilde{d}_\Omega$ satisfy*

$$\left\|\nabla \widetilde{d}_\Omega(x)\right\| \;=\; O(1) \qquad \textit{and} \qquad \left\|\nabla^2 \widetilde{d}_\Omega(x)\right\| \;=\; O\left(\frac{1}{\varepsilon}\right).$$

Observe that this is almost as good as the best query and space times for approximate polytope membership [1, 5], suffering just an additional factor $1/\sqrt{\varepsilon}$ in the space bound. Our data structure can be viewed as incorporating blending into the data structure of [1]. While we assume that the query point lies within $\Omega$, if this is not the case and $x$ is at distance at least $\varepsilon \cdot \mathrm{diam}(\Omega)$ outside, the data structure will report this. If $x$ is external to $\Omega$ but is closer than this to the boundary, it may erroneously report an answer to the query. Our focus is on the existence of the data structure, but through the use of known constructions, it can be built in time $O(n/\varepsilon^{O(d)})$, where $n$ denotes the number of facets of the polytope.

Let us remark on the bounds on the norms of the gradient and Hessian. Clearly, in any Euclidean distance field the directional derivative of the distance field is as high as 1 (when moving directly towards or away from the nearest point) and is never greater, that is, $\|\nabla d_\Omega(x)\| \leq 1$. Therefore, it is reasonable that the norm of our approximate function, $\|\nabla \widetilde{d}_\Omega(x)\|$, is $O(1)$. The following lemma shows that the $O(1/\varepsilon)$ upper bound on the norm of the Hessian is a necessity, up to constant factors. It establishes a lower bound in the context of a relative errors for approximating the distance to a discrete point set, but the result can be adapted to our context as well.

▶ **Lemma 3.** *Fix a set of points $P \subset \mathbb{R}^d$, and let $\mathcal{Q}$ be a smooth $\varepsilon$-approximate distance query structure over $P$ with the associated distance $\widetilde{d}_{\mathcal{Q}}$, for any $\varepsilon > 0$ bounding the relative error. If $|P| > 1$, then there exists a point $x \in \mathbb{R}^d$ such that $\left\|\nabla^2 \widetilde{d}_{\mathcal{Q}}(x)\right\| \geq 1/\varepsilon$.*

**Proof.** Given a function $f : \mathbb{R}^d \to \mathbb{R}$ and $\gamma \geq 0$, the assertion $\|\nabla f(a) - \nabla f(b)\| \leq \gamma$ is equivalent to saying that $\nabla f$ is $\gamma$-Lipschitz, that is, $\|\nabla f(a) - \nabla f(b)\| \leq \gamma \cdot \|a - b\|$, for all $a, b \in \mathbb{R}^d$. Letting $f := \nabla \widetilde{d}_{\mathcal{Q}}$, we will show that $f$ is $\gamma$-Lipschitz with $\gamma \geq 1/\varepsilon$.

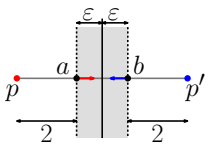


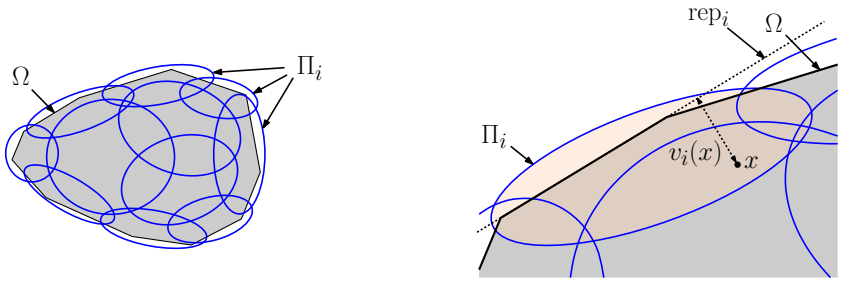

**Figure 3** Proof of Lemma 3.

Consider two sites $p$ and $p'$ such that $\|pp'\| = 2(2 + \varepsilon)$ (see Figure 3). Select points $a$ and $b$ along the segment $pp'$ on opposite sides and at distance $\varepsilon$ from the perpendicular bisector. Observe that a query point placed at any point on the open segment $pa$ must return $p$ as the answer, since otherwise the relative error would exceed $((2 + 2\varepsilon) - 2)/2 = \varepsilon$. This holds symmetrically for $p'b$. It follows that $\nabla f(a)$ and $\nabla f(b)$ are unit vectors pointing to the right and left, respectively. Hence, $\|\nabla f(a) - \nabla f(b)\|/\|a - b\| = 2/2\varepsilon = 1/\varepsilon$, as desired. ◀

The remainder of the paper is organized as follows. In the next section we present an overview of the partition-of-unity approach. In Section 3 we present an efficient data structure for answering approximate distance queries for a convex polytope $\Omega$, but without continuity. Finally, in Section 4, we combine these to obtain the desired smooth approximation.

## 2 Blending and Partition of Unity

The partition of unity is a standard mathematical tool for integrating local constructions into global ones [29, 40]. It is widely used and has applications in various disciplines [34, 35]. The approach involves a collection of *patches* $\Pi = \{\Pi_i\}$ forming a locally-finite open cover of a given domain $\Omega \subseteq \mathbb{R}^d$. The partition of unity is a set of non-negative smooth *partition functions* $\{\phi_i\}$ such that the support of $\phi_i$, denoted $\mathrm{supp}(\phi_i)$, is a subset of $\Pi_i$ (see Figure 4(a)). The name derives from the requirement that for all $x \in \Omega$, $\sum_i \phi_i(x) = 1$.




(a) (b)

**Figure 4** Patches, representatives, and the partition of unity.

In the context of distance approximation, let us assume that each patch is associated with a *local distance function* $v_i$, such that the restriction of $v_i$ to $\Pi_i$ is an $\varepsilon$-approximation to the true distance function $d_\Omega$. Concretely, each patch is associated with a *representative*,

denoted $\text{rep}_i$. For example, when approximating the distance to a discrete point set $P$, $\text{rep}_i$ may be a point $p \in P$. In our case, where $\Omega$ is a convex polytope, $\text{rep}_i$ will be chosen to be a supporting hyperplane of a facet of $\Omega$ (see Figure 4(b)). Then, $v_i(x)$ can be defined to be distance from $x$ to the associated representative,

$$v_i(x) \;=\; \text{dist}(x, \text{rep}_i). \tag{1}$$

The final approximate distance map results by taking the sum of these local distance functions over all patches weighted by the associated partition functions.

$$\widetilde{d}_\Omega(x) \;=\; \sum_i \phi_i(x) \cdot v_i(x). \tag{2}$$

Recall that the support of $\phi_i$ is limited to $\Pi_i$, so we need only compute the sum over patches containing $x$. Define the *depth* of $x$ with respect to $\Pi$, denoted $\Delta_\Pi(x)$, to be the number of patches of $\Pi$ containing $x$, and define $\Delta_\Pi = \max_x \Delta_\Pi(x)$. As in standard applications of the partition-of-unity method, we will design our patches so that $\Delta_\Pi$ is $O(1)$.

In order to enforce the condition that the functions $\phi_i$ sum to unity at any point in the domain, we will define a set of smooth, non-negative *weight functions* $\{\psi_i\}$, and then define

$$\phi_i(x) \;=\; \frac{\psi_i(x)}{\Psi(x)}, \text{ where } \Psi(x) = \sum_i \psi_i(x). \tag{3}$$

Observe that since $\widetilde{d}_\Omega(x)$ is a convex linear combination of functions, each of which is locally an $\varepsilon$-approximate distance map for $\Omega$, it follows that $\widetilde{d}_\Omega(x)$ is itself an $\varepsilon$-approximate distance map. Our construction will guarantee that there exists a positive constant $\Psi_{\min}$, such that $\Psi(x) > \Psi_{\min}$, for all $x \in \Omega$. It follows that $\phi_i(x)$ can be made as smooth as desired, being the quotient of two positive continuous functions. Assuming that the local distance approximations $\{v_i\}$ are smooth, it follows that $\widetilde{d}_\Omega$ is itself smooth, being a sum of products of pairs of continuous functions. As a 1-dimensional example, see Figure 5.

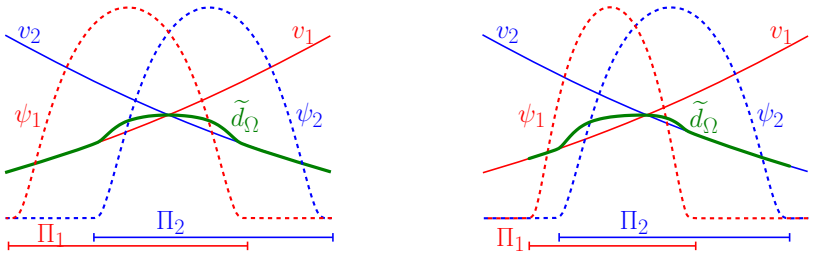



**Figure 5** Blending two distance functions $\{v_i\}$ using two overlapping intervals $\{\Pi_i\}$ with associated weight functions $\{\psi_i\}$, yielding a smooth approximation $\widetilde{d}_\Omega$ using (a) symmetric covers and (b) non-symmetric covers.

It remains to define the weight function $\psi_i$ associated with each patch. These functions depend on the patch's shape. For our application, patches will be ellipsoids, but for this introduction, let us consider the simple case of a Euclidean ball with center point $c_i$ and radius $r_i$. First, for $x \in \mathbb{R}^d$, define

$$f_i(x) \;=\; \frac{1}{r_i^2}\|x - c_i\|^2.$$

Observe that $f$ achieves its minimum value of 0 at the ball's center and grows to 1 at its boundary. To obtain a compactly-supported weight function, we use the standard technique of composing $f$ with a bump function, also known as the *standard mollifier* [39]

$$\mu(\sigma) \;=\; \begin{cases} \exp\left(-\dfrac{1}{1-\sigma^2}\right) & \text{if } |\sigma| < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

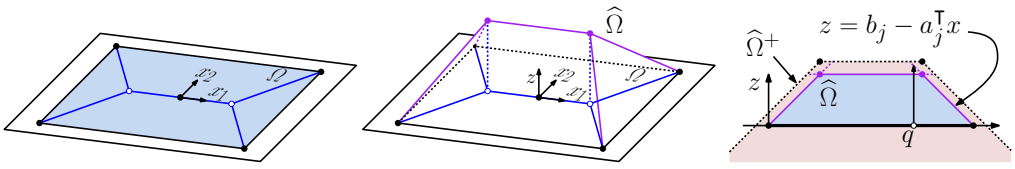
Since $\mu(0) = e^{-1}$ and $\mu(1) = 0$, we see that the weight is highest near the middle of the shape, where $f = 0$, and decays gracefully towards the boundary, where $f = 1$. It is well-known that $\mu \in C_c^\infty(\mathbb{R})$ and is non-analytic with vanishing derivatives for $|\sigma| = 1$ [39]. Therefore, we may define $\psi_i(x) = \mu(f_i(x))$.

In summary, given any query point $x$, we first determine the patches that contain it. (The number of which, $\Delta_\Pi(x)$, will be bounded by a constant.) Given the shape functions $f_i$ for each of these patches, we compute the weight functions $\psi_i$'s by applying the mollifier of Eq. (4). We then apply Eq. (3) to obtain the partition-of-unity blending functions. Finally, we apply Eqs. (1) and (2) to obtain the final smooth distance approximation. The overall space and query time are dominated by the total number of patches and the time needed to determine which patches contain the query point, respectively.

## 3 Approximating the Boundary Distance Function

The process described in the previous section is generic and can be applied in settings where the answer to the query can be expressed in terms of a covering of space by regions of low combinatorial complexity. For the sake of illustration, let us now explore how this works in the specific application of computing a smooth absolute $\varepsilon$-approximation to the boundary distance in a convex polytope $\Omega$ in $\mathbb{R}^d$. Let us assume that $\Omega$ has been scaled uniformly to unit diameter, so the absolute approximation error is $\varepsilon$.

We will employ a standard method for reducing distance approximation to a covering problem. First, let's consider the *graph* of the boundary distance function $d_\Omega$, that is, the manifold $(x, d_\Omega(x))$ in $\mathbb{R}^{d+1}$. We will use $z$ to denote coordinate values along the $(d+1)$st coordinate axis, which we will take to be directed vertically upwards in our drawings (see Figure 6(b) and (c)).




(a) (b) (c)

**Figure 6** Lifting the polytope $\Omega \subseteq \mathbb{R}^d$ to the lifted body $\widehat{\Omega} \subseteq \mathbb{R}^{d+1}$.

Assuming that the polytope $\Omega$ contains the origin in its interior, it can be represented as the intersection of a set of $n$ halfspaces in $\mathbb{R}^d$, $\Omega = \bigcap_{j=1}^n H_j$, with each $H_j$ taking the form

$$H_j \;=\; \{x \in \mathbb{R}^d \,:\, a_j^\intercal x \leq b_j\},$$

where $a_j \in \mathbb{R}^d$ is an outward-pointing unit normal vector orthogonal to $H_j$'s bounding hyperplane and $b_j \in \mathbb{R}^+$ is the distance of the bounding hyperplane from the origin. The distance of a point $x \in \Omega$ to the bounding hyperplane is the non-negative scalar $z$ such that

$x + za_j$ lies on the bounding hyperplane, that is $z = b_j - a_j^\intercal x$. The set of points lying below this surface (that is, the hypograph of the distance function) is the halfspace in $\mathbb{R}^{d+1}$ given by the linear inequality $z \leq b_j - a_j^\intercal x$. The boundary distance function is just the lower envelope (or minimization diagram [25]) of this set of halfspaces. To turn this into a bounded convex polytope, we add a horizontal ground-surface halfspace $\widehat{H}_0 = \{(x; z) : z \geq 0\}$. Define the *lifted body* $\widehat{\Omega} \subset \mathbb{R}^{d+1}$ to be

$$\widehat{\Omega} \;=\; \widehat{H}_0 \cap \bigcap_{j=1}^{n} \widehat{H}_j, \quad \text{where } \widehat{H}_j = \{(x; z) \in \mathbb{R}^{d+1} \,:\, z \leq b_j - a_j^\intercal x\} \text{ for } j \in [n].$$

Since the sides have a slope of $+1$, the diameter of $\widehat{\Omega}$ is $O(1)$.

To achieve an absolute approximation error of at most $\varepsilon$, we lift each of the upper halfspaces of $\widehat{\Omega}$ by a vertical distance of $+\varepsilon$ to obtain the resulting expanded object $\widehat{\Omega}^+$. That is, we define $\widehat{H}_j^\delta = \{(x; z) \in \mathbb{R}^{d+1} \,:\, z \leq b_j - a_j^\intercal x + \delta\}$ and $\widehat{\Omega}^+ = \bigcap_{j=1}^n \widehat{H}_j^\varepsilon$ (see Figure 6(c)). Note that the ground-surface halfspace ($\widehat{H}_0$) is not needed for $\widehat{\Omega}^+$, and hence it is unbounded. The essential features of lifting and expansion are summarized in the following lemma.

▶ **Lemma 4.** *Given a convex polytope $\Omega$ of unit diameter and any $x \in \Omega$, if a vertical ray is shot upwards from $x$ (viewed as a point in $\mathbb{R}^{d+1}$) hits a bounding hyperplane of $\widehat{\Omega}$ within $\widehat{\Omega}^+$, then the associated facet of $\Omega$ is an absolute $\varepsilon$-approximate nearest neighbor of $x$.*

The upshot is that we can base the local distance functions $v_i(x)$ (recall Eq. (1)) on the distance to the bounding hyperplane of $\Omega$ corresponding to the bounding hyperplane in the lifted body $\widehat{\Omega}$ that is hit by the vertical ray shot upwards from the query point $x$. An important feature of $\widehat{\Omega}^+$, which will be of later use (in Lemma 8), is that the distance between its boundary and that of $\widehat{\Omega}$ is at least $c\varepsilon \cdot \mathrm{diam}(\widehat{\Omega})$, for some constant $c$.

## 3.1 Macbeath Regions and Ellipsoids

Our approach to approximating $\widehat{\Omega}$ for the purpose of answering distance map queries will be based on generating a net-like covering of $\widehat{\Omega}$ based on objects called *Macbeath regions*. Macbeath regions and their variants have been widely used in convex approximation (see, e.g., [1, 4, 6–8]). In contrast to traditional covers based on subdivisions by fat objects, e.g., hypercubes, Macbeath regions naturally adapt to the shape of the object being covered. In this section we present a brief review of the salient features of Macbeath regions.

Given a convex body $\Omega$ and any point $x \in \Omega$, the *Macbeath region* at $x$ is the largest centrally-symmetric body centered at $x$ and contained within $\Omega$. It is common to apply a constant scaling factor. Formally, for $\lambda \in \mathbb{R}^+$, the *$\lambda$-scaled Macbeath region* at $x$ is

$$M_\Omega^\lambda(x) \;=\; x + \lambda((\Omega - x) \cap (x - \Omega))$$

(see Figure 7(a)). When $\Omega$ is clear from context, we will often omit the subscript. We refer to $x$ and $\lambda$ as the *center* and *scaling factor* of $M^\lambda(x)$, respectively. When $\lambda < 1$, we say $M^\lambda(x)$ is *shrunken*.

It is useful to have a low-complexity, smooth proxy for a Macbeath region. Given a Macbeath region, define its associated *Macbeath ellipsoid* $E_\Omega^\lambda(x)$ to be the maximum-volume ellipsoid contained within $M_\Omega^\lambda(x)$ (see Figure 7(b)). Clearly, this ellipsoid is centered at $x$, and $E_\Omega^\lambda(x)$ is a $\lambda$-factor scaling of $E_\Omega^1(x)$ about $x$. By John's Theorem [11]

$$E_\Omega^\lambda(x) \;\subseteq\; M_\Omega^\lambda(x) \;\subseteq\; E_\Omega^{\lambda\sqrt{d}}(x).$$

Chazelle and Matoušek showed that this ellipsoid can be computed for a convex polytope $\Omega$ in time linear in the number of its bounding halfspaces [15].

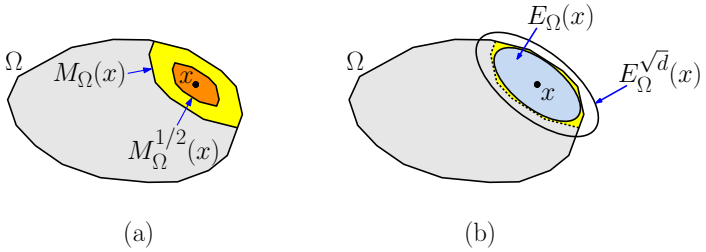



**Figure 7** (a) Macbeath regions and (b) Macbeath ellipsoids.

A fundamental property of Macbeath regions, called *expansion-containment*, states that if two shrunken Macbeath regions (or ellipsoids) overlap, then a constant-factor expansion of one contains the other. There are many formulations. The following can be found in [1].

▶ **Lemma 5** (Expansion-Containment). *Given a convex body* $\Omega \in \mathbb{R}^d$, $0 < \lambda < 1$, *let* $\beta = (3+\lambda)/(1-\lambda)$. *Then for any* $x, y \in \Omega$*:*
*(i)* $M^\lambda(x) \cap M^\lambda(y) \neq \emptyset \implies M^\lambda(y) \subseteq M^{\beta\lambda}(x)$,
*(ii)* $E^\lambda(x) \cap E^\lambda(y) \neq \emptyset \implies E^\lambda(y) \subseteq E^{\beta\lambda\sqrt{d}}(x)$.

## 3.2 Approximation through Covering

Our approach to computing an $\varepsilon$-approximation to the boundary distance function within a convex polytope $\Omega$ in $\mathbb{R}^d$ utilizes Macbeath regions to cover the lifted body $\widehat{\Omega}$ in $\mathbb{R}^{d+1}$. Recall our assumption that $\Omega$ has been scaled to unit diameter, and hence the lifted body $\widehat{\Omega}$ also has unit diameter. Given this scaling, our objective is to answer vertical ray-shooting queries up to an absolute error of at most $\varepsilon$, (see Lemma 4).

Before presenting our solution, let us recall some known results for convex approximation. Given a convex body $\Omega$ of unit diameter and $\varepsilon > 0$, an *$\varepsilon$-approximate polytope membership query* is given a query point $q$ and returns positive answer if $q$ lies within $\Omega$, a negative answer if $q$ lies at distance more than $\varepsilon$ from $\Omega$, and otherwise, it may give either answer. Arya *et al.* presented an efficient data structure for answering approximate membership queries [7]. Later, Abdelkader and Mount [1] presented a simpler approach with the same space and query times, as described in the following lemma. We will employ a variant of the latter data structure.

▶ **Lemma 6.** *Given a convex polytope* $\Omega \in \mathbb{R}^d$, *there exists a data structure that can answer absolute* $\varepsilon$*-approximate polytope membership queries for* $\Omega$ *in time* $O(\log(1/\varepsilon))$ *and storage* $O(1/\varepsilon^{(d-1)/2})$.

In order to apply this data structure for our purposes, we will need to delve a bit deeper into how it works. Our application of this structure will be in the lifted space $\mathbb{R}^{d+1}$, but let us describe it now for an arbitrary convex body $\Omega$ in $\mathbb{R}^d$. Given a non-negative parameter $\delta$, define the *expanded body* $\Omega_\delta$ to be a convex set such that $\Omega \subseteq \Omega_\delta$, and the minimum distance between their boundaries of is at least $\delta$.

Next, we define the notion of a *Macbeath-based Delone set* of $\Omega$ relative to $\Omega_\delta$. This structure is parameterized by two constants $0 < \lambda_p < \lambda_c < 1$, called the *packing* and *covering* constants, respectively (which may depend on the dimension $d$). Given any point $x \in \Omega$, define the *covering ellipsoid* $E'_\delta(x) = E^{\lambda_c}_{\Omega_\delta}(x)$, that is, a Macbeath ellipsoid centered at $x$ with scaling factor $\lambda_c$ defined with respect to the outer body $\Omega_\delta$. Define the *packing ellipsoid* $E''_\delta(x)$ analogously, but with a scaling factor of $\lambda_p$. A *Macbeath-based Delone set* for $\Omega$

relative to $\Omega_\delta$ is any maximal set of points $X \subset \Omega$, such that the packing ellipsoids $E''_\delta(x)$ centered at these points are pairwise disjoint. Abdelkader and Mount [1] showed that, by standard properties of Macbeath regions, constants $\lambda_p$ and $\lambda_c$ can be chosen such that $X$ has the following properties:

**(i)** The union of the covering ellipsoids $E'_\delta(x)$ over all $x \in X$ covers the original body, $\Omega$,
**(ii)** For each $x \in X$, $E'_\delta(x)$ is contained within the expanded body, $\Omega_\delta$,
**(iii)** The number of ellipsoids $E'_\delta$ that contain any point $x$ is $O(1)$,
**(iv)** $|X| = O(1/\delta^{(d-1)/2})$.

Note that the constant factors hidden in the $O$-notation depend on the dimension $d$.

To turn this into an approximate search structure, a layered DAG is constructed as follows. For $i \geq 0$, let $\delta_i = 2^i\delta$. Construct a series of such Delone sets, $X_0, X_1, \ldots, X_m$ where $X_i$ is any Macbeath-based Delone set of $\Omega$ with respect to $\Omega_{\delta_i}$. As $i$ increases, the expanded body grows larger, and hence the Macbeath ellipsoids also grow larger. But since they need only cover the original body $\Omega$, their size, $|X_i|$, decreases with $i$. The final layer $\ell_m$ is defined to be the smallest integer such that $|X_{\ell_m}| = 1$. (It can be shown that $\ell_m = O(\text{diam}(\Omega)) = O(1)$, which implies that $m = O(\log(1/\delta))$.) The leaves of the DAG correspond to the covering ellipsoids $E'_{\delta_0}$ centered at the points of $X_0$. The root is corresponds to the covering ellipsoid $E'_{\delta_{\ell_m}}$ associated with the single point of $X_{\ell_m}$ (see Figure 8). Finally, the nodes of level $i$ are connected to nodes at level $i-1$ whenever their associated $E'$ ellipsoids overlap. Abdelkader and Mount [1] showed the following:

- The DAG has $O(\log(1/\varepsilon))$ layers.
- The out-degree of any node in the DAG is $O(1)$.
- The total number of nodes in the DAG is $O(1/\delta^{(d-1)/2})$.

Lemma 6 follows by applying a natural search process which simply descends from the root to any leaf in the DAG, always visiting a node whose $E'$ ellipsoid contains the query point. If a query point $x \in \Omega$, the search will succeed in finding a leaf-level ellipsoid $E'$ that contains this point.

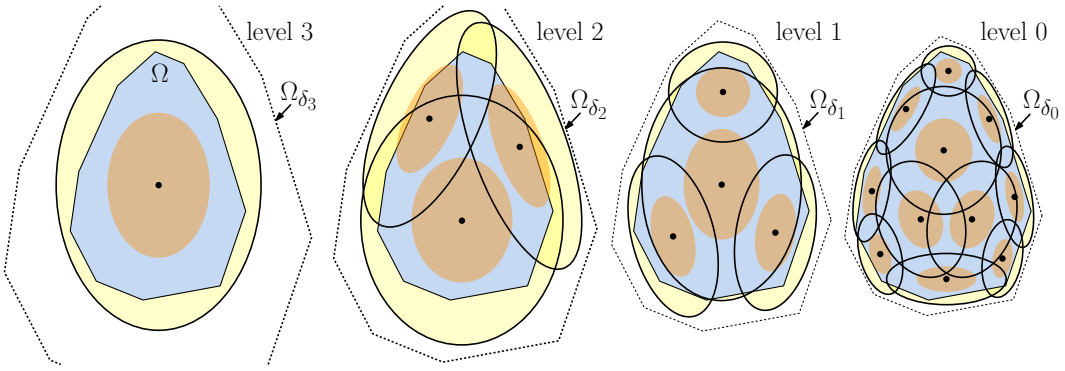



**Figure 8** Layers of the Macbeath-based Delone set data structure.

## 3.3 Approximation through Vertical Ray Shooting

We can now explain how to construct the smooth boundary distance approximation for $\Omega$. First, we construct the lifted body $\widehat{\Omega}$, as described in Section 3. Given a query point $x \in \Omega$, its distance to the boundary is determined by the height of the point on $\partial\widehat{\Omega}$ hit by an upward-directed vertical ray shot from $x$ in $\mathbb{R}^{d+1}$. To apply the hierarchical search, let $\delta = \varepsilon$,

and for any $\ell \geq 0$, define $\widehat{\Omega}_{\delta_\ell}$ to be the unbounded convex set that results by translating all the upper halfspaces bounding $\widehat{\Omega}$ up by distance $\delta_\ell = 2^\ell \varepsilon$. That is, $\widehat{\Omega}_{\delta_\ell} = \bigcap_{j=1}^{n} \widehat{H}_j^{\delta_\ell}$. Observe that $\widehat{\Omega}_{\delta_0} = \widehat{\Omega}^+$, the $\varepsilon$-expanded body.

We can apply the data structure described in Lemma 6 to these bodies in $\mathbb{R}^{d+1}$. For any level of the structure, we say that a covering ellipsoid is a *top ellipsoid* if there exists $x \in \Omega$, such that this ellipsoid has the highest intersection point with the vertical ray directed up from $x$ among all the ellipsoids in the cover. Because the covering ellipsoids cover $\widehat{\Omega}$ it follows that the union of the top ellipsoids, when projected vertically onto $\mathbb{R}^d$, covers the original body $\Omega$.

To answer vertical ray-shooting queries using our hierarchy, we traverse the hierarchy of ellipsoids, but whenever we descend a level in the DAG structure, among all the nodes whose covering ellipsoid $E'$ intersects the vertical line segment passing through $x$, we visit the one having the highest point of intersection with the ray (see Figure 9). It was shown in [1] that the number of ellipsoids that need to be considered is $O(1)$. Therefore, in time proportional to the number of levels, which is $O(\log(1/\varepsilon))$, we can find the top ellipsoid at the leaf level traversed by the vertical ray.

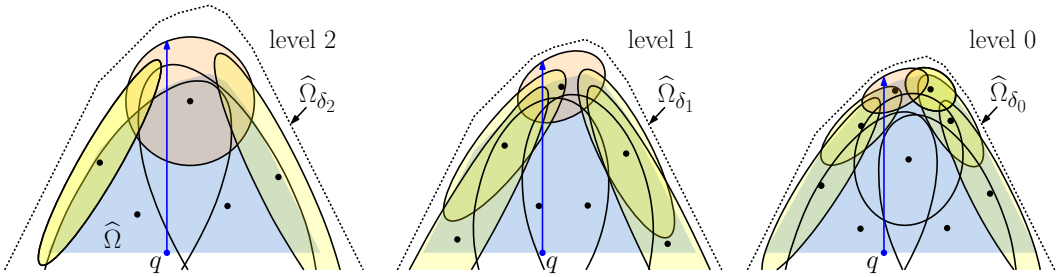



**Figure 9** Vertical ray-shooting in the hierarchical structure.

Furthermore, because all these ellipsoids lie within $\widehat{\Omega}_{\delta_0}$, the terminus of the vertical ray lies within the vertical gap of length $\varepsilon$ between $\widehat{\Omega}$ and $\widehat{\Omega}^+$, as required in Lemma 4. Finally, through an appropriate adjustment of the scaling factors $\lambda_p$ and $\lambda_c$, we can apply the same analysis as in Arya *et al.* [4, Lemma 3.5] to find a witness hyperplane that serves as the representative for all vertical rays passing through this ellipsoid. Since the construction is performed in $\mathbb{R}^{d+1}$, the space required is $O(1/\varepsilon^{d/2})$. This implies that, even ignoring continuity, we can answer $\varepsilon$-approximate boundary distance queries efficiently.

▶ **Theorem 7.** *Given a convex polytope $\Omega \in \mathbb{R}^d$, there exists a data structure that can answer absolute $\varepsilon$-approximate boundary distance queries (without continuity guarantees) for $\Omega$ in time $O(\log(1/\varepsilon))$ and storage $O(1/\varepsilon^{d/2})$.*

There are a couple of additional properties, which will be useful for the task of computing smooth distance approximations. First, because the boundaries of $\widehat{\Omega}$ and $\widehat{\Omega}^+$ are separated by a vertical distance of $\varepsilon$, we can infer that the Macbeath ellipsoids cannot be too skinny.

▶ **Lemma 8.** *Each of the covering Macbeath ellipsoids in the data structure of Theorem 7 contains a Euclidean ball at its center of radius at least $c\varepsilon$, for some constant $c$ (depending on $\lambda_c$ and $d$).*

**Proof.** The centers lie within $\widehat{\Omega}$, and Macbeath regions are defined with respect to $\widehat{\Omega}^+$ whose boundary has a vertical separation of $\varepsilon$. Because the distance function is 1-Lipschitz continuous, the sides of both of these bodies have a slope (with respect to vertical) of at

most 1. Hence, the distance from any point in $\widehat{\Omega}$ to the boundary of $\widehat{\Omega}^+$ is at least $\varepsilon/\sqrt{d}$. Therefore, the scale-1 Macbeath region contains a ball of this radius at its center. The covering Macbeath region has a ball of radius $\varepsilon(\lambda_c/\sqrt{d})$. The John ellipsoid contains a ball of radius $\varepsilon(\lambda_c/\sqrt{d})/\sqrt{d} = \varepsilon(\lambda_c/d)$. Setting $c = \lambda_c/d$, completes the proof. ◀

Second, from the properties of the Macbeath-based Delone set, each point of $\widehat{\Omega}$ is covered by only a constant number of the covering ellipsoids at the leaf level. While this does not necessarily hold for the vertical projections of these ellipsoids, it does hold when we restrict attention to the top ellipsoids. Let $\Delta_\Pi$ denote maximum number of ellipsoids that may contain any given point $x \in \Omega$.

▶ **Lemma 9.** *The blending patches* $\Pi$ *resulting from the vertical projections of the top covering ellipsoids have constant depth, that is,* $\Delta_\Pi = O(1)$.

Before giving the proof, we establish a useful technical lemma. Let us begin with some notation. Given a concave function $f : \mathbb{R}^d \to \mathbb{R}$, its *hypograph*, denoted $f^-$, is the set of points in $\mathbb{R}^{d+1}$ lying on or below the function. Clearly, $f^-$ is a convex set. For any point $x \in f^-$, define the *ray distance* of $x$ with respect to $f^-$, denoted $\text{ray}_{f^-}(x)$ to be the length of a ray shot upwards from $x$ to the boundary of $f^-$. For any $x \in f^-$ and $\lambda \geq 0$, define $M^\lambda_{f^-}(x)$ as the $\lambda$-scaled Macbeath region relative to $f^-$. We omit explicit references to $f^-$ in subscripts when it is clear from context.

▶ **Lemma 10.** *Given concave* $f : \mathbb{R}^d \to \mathbb{R}$, $x \in f^-$ *and* $\lambda \geq 0$, *for all* $y \in M^\lambda_{f^-}(x)$,

$$(1-\lambda)\cdot \text{ray}_{f^-}(x) \;\leq\; \text{ray}_{f^-}(y) \;\leq\; (1+\lambda)\cdot \text{ray}_{f^-}(x).$$

**Proof.** To simplify notation, let $r_x = \text{ray}_{f^-}(x)$ and $r_y = \text{ray}_{f^-}(y)$. To prove the upper bound, let $x'$ denote the intersection of the vertical ray through $x$ with the boundary of $f^-$ (see Figure 10(a)). Consider a supporting hyperplane $h_0$ for $f^-$ passing through $x'$. Let $h_1$ be the parallel supporting hyperplane passing through $x$, and let $h_2$ be the parallel supporting hyperplane along the lower side of $M^\lambda(x)$. Clearly, the vertical distance between $h_0$ and $h_1$ is $r_x$, and the vertical distance between $h_1$ and $h_2$ is $\lambda r_x$. Since $M^\lambda(x)$ lies entirely above $h_2$, it follows that the vertical segment defining $r_y$ lies entirely below $h_0$ and above $h_2$, which implies that $r_y \leq r_x + \lambda r_x = (1+\lambda)r_x$, as desired.

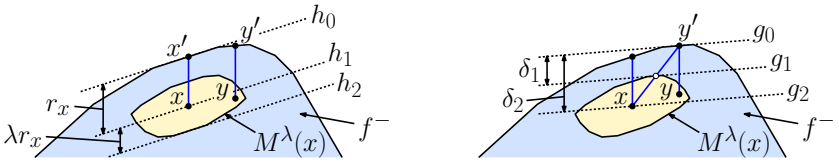



**Figure 10** Proof of Lemma 10.

To prove the lower bound, let $y'$ denote the point where the vertical ray through $y$ intersects the boundary of $f^-$ (see Figure 10(b)). Let $g_0$ denote a supporting hyperplane for $f^-$ passing through $y'$. Let $g_1$ be the upper parallel supporting hyperplane for $M^\lambda(x)$, and let $g_2$ be the parallel hyperplane passing through $x$. Let $\delta_1$ denote the vertical distance between $g_0$ and $g_1$, and let $\delta_2$ denote the vertical distance between $g_0$ and $g_2$. By definition of the Macbeath region, we have $\delta_2 - \delta_1 = \lambda\delta_2$, or equivalently $\delta_1 = (1-\lambda)\delta_2$. Clearly, $y$ lies below $g_1$, and so $r_y \geq \delta_1$. Since all of $f^-$ (including $x'$) lies below $g_0$, we have $r_x \leq \delta_2$. Therefore, $r_y \geq \delta_1 = (1-\lambda)\delta_2 \geq (1-\lambda)r_x$, as desired. ◀

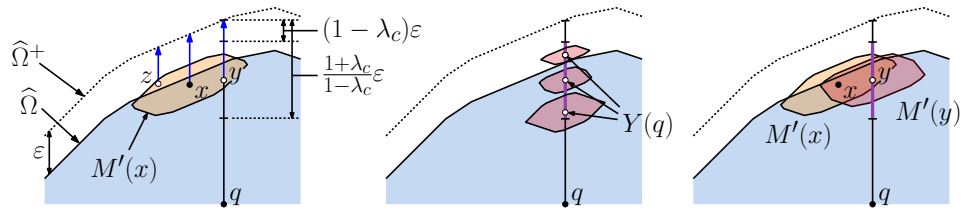
**Proof of Lemma 9.** Recall that constants $\lambda_c$ and $\lambda_p$ are the so called *covering* and *packing* scale factors used in our construction. Given a point $x \in \Omega^+$, let $M'(x)$ and $M''(x)$ denote respectively the covering ($\lambda_c$-scaled) and packing ($\lambda_p$-scaled) Macbeath regions centered at $x$ with respect to the expanded body $\widehat{\Omega}^+$. Define $E'(x)$ and $E''(x)$ analogously for Macbeath ellipsoids.

Recall that our construction is based on a maximal point set $X \subset \widehat{\Omega}$ such that the packing ellipsoids $E''(x)$ are disjoint for $x \in X$ and $E'(x)$ cover $\widehat{\Omega}$. Given any $q \in \Omega$, let $X(q) \subseteq X$ denote the set of top covering ellipsoids $E'(x)$ whose vertical projection contains $q$. Equivalently, $x \in X(q)$ if the vertical ray passing through $q$ intersects $E'(x)$. It suffices to show that for any $q \in \Omega$, $|X(q)| = O(1)$.

For any $x \in \widehat{\Omega}^+$, define $\mathrm{ray}(x)$ to be the length of a vertical ray shot from $x$ up to the boundary of $\widehat{\Omega}^+$. By definition of a top ellipsoid, for any $x \in X(q)$, there exists a point $z \in E'(x)$ such that $\mathrm{ray}(z) \leq \varepsilon$. Since $E'(x) \subseteq M'(x)$, we have $z \in M'(x)$ (see Figure 11(a)). Thus, by Lemma 10 with $\widehat{\Omega}^+$ playing the role of $f^-$ and $z$ playing the role of $y$, it follows that $\mathrm{ray}(x) \leq \mathrm{ray}(z)/(1-\lambda_c) \leq \varepsilon/(1-\lambda_c)$. Applying the lemma again, it follows that for any other point $y \in E'(x)$, we have

$$\mathrm{ray}(y) \;\leq\; (1+\lambda_c)\cdot \mathrm{ray}(x) \;\leq\; \frac{1+\lambda_c}{1-\lambda_c}\varepsilon.$$

Also, because $x \in \widehat{\Omega}$, we have $\mathrm{ray}(x) \geq \varepsilon$, implying again by Lemma 10 that $\mathrm{ray}(y) \geq (1-\lambda_c)\varepsilon$. In summary, for each $x \in X(q)$, there exists a point $y$ along the vertical ray shot up from $q$ such that $(1-\lambda_c)\varepsilon \leq \mathrm{ray}(y) \leq \frac{1+\lambda_c}{1-\lambda_c}\varepsilon$.




(a) (b) (c)

**Figure 11** Proof of Lemma 9.

Let $Y(q)$ be any maximal set of points along the vertical line through $q$ that have ray distances in the interval $\left[(1-\lambda_c), \frac{1+\lambda_c}{1-\lambda_c}\right]\varepsilon$ and whose packing Macbeath regions are pairwise disjoint (see Figure 11(b)). Each such Macbeath region covers an interval of length at least $\lambda_p(1-\lambda_c)\varepsilon$. By a standard packing argument, there are at most a constant $c'$ (depending on $\lambda_p$ and $\lambda_c$) of such Macbeath regions, and their covering Macbeath regions cover this subsegment of the vertical line.

Now, associate each point $x \in X(q)$ with any one of the points of $y \in Y(q)$, such that $M'(x) \cap M'(y) \neq \emptyset$ (see Figure 11(c)). By the prior observations, such a point of $y$ exists for each $x$. By expansion containment (Lemma 5), a constant factor expansion of $M'(y)$ contains $M'(x)$ and vice versa. Therefore, the volumes of these bodies are equal up to constant factors (depending on $\lambda_c$ and the dimension $d$). Because $\lambda_p$ and $\lambda_c$ are both constants, the volumes of $M''(x)$ and $M''(y)$ are related by constant factors. Thus, by a straightforward packing argument, the disjointness of the $M''(x)$ Macbeath regions implies that the number of $x \in X(q)$ that are associated with any $y \in Y(q)$ is bounded above by some constant $c''$. Thus, we have $|X(q)| \leq c'c'' = O(1)$, as desired. ◀

## 4 Putting it Together

We can now explain how to combine the results of the previous section with the partition-of-unity method from Section 2, to obtain the final smooth distance approximation.

The set of patches $\Pi = \{\Pi_i\}$ used in blending consist of the vertical projections of all the top ellipsoids from level-0 of the vertical ray-shooting data structure. Each ellipsoidal patch $\Pi_i$ is represented by its center $c_i \in \mathbb{R}^d$ and a positive-definite matrix $M_i$ such that

$$\Pi_i \;=\; \{x \in \mathbb{R}^d : f_i(x) \le 1\}, \qquad \text{where } f_i(x) \;=\; (x - c_i)^\intercal M_i (x - c_i). \tag{5}$$

Recalling the definition of the standard mollifier $\mu$ from Eq. (4), we define

$$\psi_i(x) \;=\; \mu(f_i(x)). \tag{6}$$

Given these weight functions, we apply Eq. (3) to obtain the blending function $\phi_i(x)$ for each patch. Recall that each of the top ellipsoids $\Pi_i$ is associated with a representative of the upper envelope of $\widehat{\Omega}$ in the form of a halfspace $H_i = \{x \in \mathbb{R}^d : a_i^\intercal x \le b_i\}$. As mentioned in Section 3, the associated local distance function is $v_i(x) = b_i - a_i^\intercal x$ (Eq. (1)).

Given a query point $x$, we use the vertical ray-shooting data structure to determine the patches $\Pi_i$ containing it. By Lemma 9, there are a constant number of them. We apply Eq. (2) to blend together the local distance functions to obtain the final distance approximation, $\widetilde{d}_\Omega(x)$. The space and query time are dominated by the complexity bounds for the ray-shooting data structure, given in Lemma 7. This establishes the correctness and complexity bounds of Theorem 2. The bounds on the norms of the gradient and Hessian are rather technical and will appear in the full version.

## 5 Concluding Remarks

In this paper, we have taken first steps towards designing data structures for approximately answering geometric distance queries approximately, while more faithfully preserving properties of the underlying distance functions. Existing data structures based on computing approximate nearest neighbors suffer from discontinuities in the resulting distance field, which is undesirable in many applications. We have presented a general method for achieving smoothness by combining a traditional (discontinuous) method with blending, and we have illustrated the technique in the concrete application of approximating (in terms of absolute errors) the distance field to the boundary, induced within a convex polytope $\Omega$ in $\mathbb{R}^d$. Our data structure is efficient in the sense that it nearly matches the best asymptotic space and time bounds for the simpler problem of approximately determining membership within the polytope (being suboptimal by a factor of $1/\sqrt{\varepsilon}$ in the space). We have also presented bounds on the norms of the gradient (first derivative) and Hessian (second derivative) of the approximation.

There are a number of interesting open problems that remain. The first is applying this method to more approximate nearest neighbor search applications. We have done this for a discrete set of points in $\mathbb{R}^d$, which we plan to publish in a future paper. While our results nearly matching the best known complexity bounds for $\varepsilon$-approximate nearest neighbor searching, the technical issues are quite involved. The method can be applied to other query problems where the answer is naturally associated with a continuous field. Examples include penetration depth in collision detection [47], distance oracles in robotics and autonomous navigation [44], and novel-view synthesis using parametric radiance fields [20].

While our approach produces a smooth approximation, there are other properties of distance fields that would be useful to preserve. One shortcoming of our method is that it can produce spurious local minima in the approximate distance field. An interesting question is whether our approach can be modified to eliminate these minima. We anticipate interesting connections to the literature on vector field design [43] and mode finding [28].

## References


1 A. Abdelkader and D. M. Mount. Economical Delone sets for approximating convex bodies. In *Proc. 16th Scand. Workshop Algorithm Theory*, pages 4:1–4:12, 2018. doi:10.4230/LIPIcs.SWAT.2018.4.

2 R. Al-Aifari, I. Daubechies, and Y. Lipman. Continuous procrustes distance between two surfaces. *Commun. Pure and Appl. Math.*, 66:934–964, 2013. doi:10.1002/cpa.21444.

3 N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. *Discrete Comput. Geom.*, 22:481–504, 1999. doi:10.1007/PL00009475.

4 R. Arya, S. Arya, G. D. da Fonseca, and D. M. Mount. Optimal bound on the combinatorial complexity of approximating polytopes. *ACM Trans. Algorithms*, 18(4):1–29, 2022. doi:10.1145/3559106.

5 S. Arya, G. D. da Fonseca, and D. M. Mount. Near-optimal $\varepsilon$-kernel construction and related problems. In *Proc. 33rd Internat. Sympos. Comput. Geom.*, pages 10:1–15, 2017. doi:10.4230/LIPIcs.SoCG.2017.10.

6 S. Arya, G. D. da Fonseca, and D. M. Mount. On the combinatorial complexity of approximating polytopes. *Discrete Comput. Geom.*, 58(4):849–870, 2017. doi:10.1007/s00454-016-9856-5.

7 S. Arya, G. D. da Fonseca, and D. M. Mount. Optimal approximate polytope membership. In *Proc. 28th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 270–288, 2017. doi:10.1137/1.9781611974782.18.

8 S. Arya, G. D. da Fonseca, and D. M. Mount. Economical convex coverings and applications. In *Proc. 34th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1834–1861, 2023. doi:10.1137/1.9781611977554.ch70.

9 S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *J. Assoc. Comput. Mach.*, 45(6):891–923, 1998. doi:10.1145/293347.293348.

10 F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing Co., Inc., 1st edition, 2013.

11 K. Ball. An elementary introduction to modern convex geometry. In S. Levy, editor, *Flavors of Geometry*, pages 1–58. Cambridge University Press, 1997. (MSRI Publications, Vol. 31).

12 R. G. Bartle and D. R. Sherbert. *Introduction to Real Analysis*. Wiley, 3rd edition, 1999.

13 J. Bloomenthal, C. Bajaj, J. Blinn, B. Wyvill, M.-P. Cani, A. Rockwood, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

14 T. Brochu, E. Edwards, and R. Bridson. Efficient geometrically exact continuous collision detection. *ACM Trans. Graph.*, 31(4):96:1–96:7, 2012. doi:10.1145/2185520.2185592.

15 B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996. doi:10.1006/jagm.1996.0060.

16 J. Chibane, A. Mir, and G. Pons-Moll. Neural unsigned distance fields for implicit function learning. In *Proc. 34th Internat. Conf. Neural Inf. Proc. Syst.*, 2020. doi:10.48550/arXiv.2010.13938.

17 T. Culver, J. Keyser, and D. Manocha. Accurate computation of the medial axis of a polyhedron. In *Proc. Fifth ACM Symp. Solid Modeling and Applications*, SMA '99, pages 179–190, 1999. doi:10.1145/304012.304030.

18 H. Edelsbrunner and J. Harer. *Computational topology: An introduction*. American Mathematical Soc., 2010.

**19** D. Eppstein and J. Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. In *Proc. 14th Annu. Sympos. Comput. Geom.*, pages 58–67, 1998. doi:10.1145/276884.276891.

**20** S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proc. IEEE/CVF Conf. Comput. Vis. Patt. Recog. (CVPR)*, pages 5501–5510, 2022. doi:10.1109/CVPR52688.2022.00542.

**21** A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, 1999.

**22** A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit geometric regularization for learning shapes. In *Internat. Conf. Machine Learning*, pages 3789–3799, 2020.

**23** S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

**24** S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theo. of Comput.*, 8:321–350, 2012.

**25** S. Har-Peled and N. Kumar. Approximating minimization diagrams and generalized proximity search. *SIAM J. Comput.*, 44:944–974, 2015.

**26** S. Har-Peled, N. Kumar, D. M. Mount, and B. Raichel. Space exploration via proximity search. *Discrete Comput. Geom.*, 56:357–376, 2016. doi:10.1007/s00454-016-9801-7.

**27** P. Kopp, E. Rank, V. M. Calo, and S. Kollmannsberger. Efficient multi-level hp-finite elements in arbitrary dimensions. *Comput. Meth. Appl. Mech. Eng.*, 401, 2022. doi:10.1016/j.cma.2022.115575.

**28** J. C. H. Lee, J. Li, C. Musco, J. M. Phillips, and W. M. Tai. Finding an approximate mode of a kernel density estimate. In *Proc. 29th Annu. European Sympos. Algorithms*, pages 61:1–61:19, 2021. doi:10.4230/LIPIcs.ESA.2021.61.

**29** J. M. Lee. *Smooth Maps*, pages 30–59. Springer New York, 2003.

**30** Y. Lipman. Phase transitions, distance functions, and implicit neural representations. In *Proc. 38th Internat. Conf. Machine Learning*, pages 6702–6712, 2021.

**31** R. Lopez-Padilla, R. Murrieta-Cid, and S. M. LaValle. Optimal gap navigation for a disc robot. In E. Frazzoli, T. Lozano-Perez, N. Roy, and D. Rus, editors, *Algorithmic Foundations of Robotics X*, pages 123–138, 2013. doi:10.1007/978-3-642-36279-8_8.

**32** G. L. Marchetti, V. Polianskii, A. Varava, F. T. Pokorny, and D. Kragic. An efficient and continuous Voronoi density estimator. In *Proc. 26th Internat. Conf. Artif. Intel. Stat.*, pages 4732–4744, 2023. doi:10.48550/arXiv.2210.03964.

**33** P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17:179–184, 1970.

**34** J. M. Melenk and I. Babuška. The partition of unity finite element method: Basic theory and applications. *Computer Methods in Applied Mechanics and Engineering*, 139:289–314, 1996.

**35** Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22:463–470, 2003.

**36** S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer New York, 2003.

**37** J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, pages 165–174, 2019.

**38** N. Sharp and A. Jacobson. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. *ACM Trans. Graph.*, 41:1–16, 2022. doi:10.1145/3528223.3530155.

**39** R. E. Showalter. *Hilbert space methods in partial differential equations*. Dover Publications, 2011.

40 E. M. Stein. *Singular Integrals and Differentiability Properties of Functions*. Princeton Mathematical Series (PMS-30). Princeton University Press, 1970. URL: https://www.jstor.org/stable/j.ctt1bpmb07.

41 A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea. 3D Skeletons: A State-of-the-Art Report. *Computer Graphics Forum*, 35(2):573–597, 2016. doi:10.1111/cgf.12865.

42 K. Tiwari, B. Sakcak, P. Routray, Manivannan M., and S. M. LaValle. Visibility-inspired models of touch sensors for navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

43 A. Vaxman, M. Campen, O. Diamanti, D. Panozzo, D. Bommes, K. Hildebrandt, and M. Ben-Chen. Directional field synthesis, design, and processing. In *Computer Graphics Forum*, volume 35, pages 545–572, 2016.

44 V. J. Wei, R. C.-W. Wong, C. Long, D. M. Mount, and H. Samet. Proximity queries on terrain surface. *ACM Trans. Database Syst.*, 47:1–59, 2022. doi:10.1145/3563773.

45 A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Trans. Robotics*, 23(1):151–157, 2007.

46 N. Zander, H. Bériot, C. Hoff, P. Kodl, and L. Demkowicz. Anisotropic multi-level hp-refinement for quadrilateral and triangular meshes. *Finite Elem. Anal. Design*, 203, 2022. doi:10.1016/j.finel.2021.103700.

47 X. Zhang, Y. J. Kim, and D. Manocha. Continuous penetration depth. *Computer-Aided Design*, 46:3–13, 2014.

# Reconfiguration of Polygonal Subdivisions via Recombination

**Hugo A. Akitaya**
University of Massachusetts Lowell, MA, USA

**Andrei Gonczi**
Tufts University, Medford, MA, USA

**Diane L. Souvaine**
Tufts University, Medford, MA, USA

**Csaba D. Tóth**
California State University Northridge, Los Angeles, CA, USA
Tufts University, Medford, MA, USA

**Thomas Weighill**
University of North Carolina Greensboro, NC, USA

## Abstract

Motivated by the problem of redistricting, we study area-preserving reconfigurations of connected subdivisions of a simple polygon. A connected subdivision of a polygon $\mathcal{R}$, called a *district map*, is a set of interior disjoint connected polygons called *districts* whose union equals $\mathcal{R}$. We consider the *recombination* as the reconfiguration move which takes a subdivision and produces another by merging two adjacent districts, and by splitting them into two connected polygons of the same area as the original districts. The *complexity* of a map is the number of vertices in the boundaries of its districts. Given two maps with $k$ districts, with complexity $O(n)$, and a perfect matching between districts of the same area in the two maps, we show constructively that $(\log n)^{O(\log k)}$ recombination moves are sufficient to reconfigure one into the other. We also show that $\Omega(\log n)$ recombination moves are sometimes necessary even when $k = 3$, thus providing a tight bound when $k = 3$.

**2012 ACM Subject Classification** Social and professional topics → Social engineering attacks; Theory of computation → Computational geometry; Theory of computation → Nonconvex optimization

**Keywords and phrases** configuration space, gerrymandering, polygonal subdivision, recombination

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.6

**Related Version** *Full Version*: https://arxiv.org/abs/2307.00704

**Funding** *Csaba D. Tóth*: Research funded in part by NSF awards DMS-1800734 and DMS-2154347.
*Thomas Weighill*: Research funded in part by NSF award OIA-1937095.

## 1 Introduction

We consider the problem of *redistricting* – the partition of a geographic domain into disjoint districts. In particular, we consider the case when these districts are required to be connected and of roughly equal population. These criteria are typically enforced in political redistricting, wherein each district elects one or more representatives to serve on a governing body, a canonical example being Congressional districts in the United States. Even under these restrictions, the space of possible redistricting plans for a typical domain is intractably vast, making it difficult to sample from this space. Recently, algorithms for generating large samples of plans have made it possible to find the neutral baseline for a particular state, which in turn can be used to detect and describe gerrymanders (i.e., unfair maps) [9, 8, 10, 13, 14, 17, 18].

The most common and successful sampling algorithms for redistricting are Markov chains that perform a sequence of reconfiguration moves on an initial map. The most prominent reconfiguration move is the *recombination* or *ReCom* move (see Figure 1), which is a move that modifies two adjacent districts while maintaining population balance and connectivity [13, 14]. In order to properly sample from the space of redistricting plans, we should require that any feasible redistricting plan can be reached from the initial map by a finite sequence of ReCom moves. That is, we want to positively answer the *reachability* question for this reconfiguration move; in the language of Markov chains this would be to prove that any chain built on the ReCom move is *irreducibile*.

Historically, most redistricting algorithms have operated on a discretized version of the geographic domain. In this framework, a district map is modeled as a vertex partition of an adjacency graph [14, 24]. This is natural since population data is only available at the level of fixed geographic units, such as Census blocks in the case of the United States. The ReCom algorithm fits within this framework, and current versions all use a spanning tree method on the adjacency graph to perform the ReCom move. Unfortunately, it is easy to construct small pathological examples of graphs for which ReCom reachability fails. Moreover, even determining whether two plans can be connected via a sequence of ReCom moves is PSPACE-complete [3] for general (planar) graphs.

A reasonable but unproven hypothesis is that for real-world adjacency graphs representing sufficiently fine discretizations of the geographic domain, we will indeed have reachability. A general theorem covering all adjacency graphs of interest seems beyond reach, which has led to a search for intermediate results. One direction of investigation is to allow a large class of graphs but relax the population balance constraint considerably; in such cases theoretical results are possible [2, 3] (see Related Work below). Reachability on grid graphs or triangular lattices is an active area of research but as of yet without concrete results.

In this paper, we return to the original hypothesis – that sufficient discretization leads to reachability – to motivate our result. Instead of modeling redistricting plans as graph partitions, we adopt a continuous model where the districts are connected polygons of equal population which partition a polygonal domain. Note that sampling algorithms based on this model do exist in the literature, most notably the power diagram method in [11], but these algorithms are not Markov chains and require an extra refinement step to go from polygonal partitions to partitions that respect the geographic units.

In our continuous model, we are able to establish reachability for the ReCom move – that is, any two polygonal partitions can be connected by finitely many ReCom steps that merge and resplit adjacent polygonal districts. The implication is that given two real-world redistricting plans, a sufficiently fine discretization of the geographic domain allows a finite sequence of ReCom moves (on the adjacency graph) to connect them. In practice this could mean that a particular map is not reachable from the initial map when considering voting precincts as geographic units, but could become reachable when working with Census blocks.

**Related Work.** In the discrete setting, the context for the reachability problem consists of a graph $G$ with $n$ nodes, a number of districts $k$ and a *slack* $\varepsilon \geq 0$. Valid partitions are defined as partitions of $V(G)$ into $k$ non-empty subsets (called *districts*) that each induce connected subgraphs such that the number of vertices in each district lies in the interval $[(1-\varepsilon)\frac{n}{k}, (1+\varepsilon)\frac{n}{k}]$. Two common reconfiguration moves on the space of valid partitions are the switch move and ReCom move. A *switch move* [15, 21] consists of reassigning a single node to a new district. Using the switch move allows one to construct a Markov chain on the space of valid partitions with easily computable transition probabilities. A

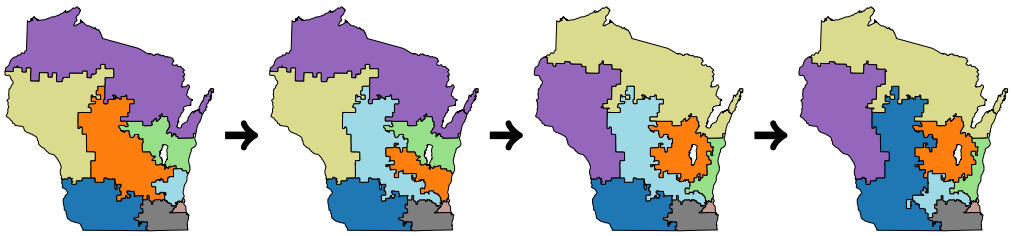


**Figure 1** A sequence of three recombination moves on the state of Wisconsin. At each step, two districts are merged and split again. The reachability problem is to determine whether any map can be reached from any other by a finite sequence of such steps.

Metropolis-Hastings weighting can then be used to ensure that the chain samples (in the limit) from any desired distribution on the space of valid partitions. Crucially, however, this relies on the assumption that the state space is connected, i.e., that any two partitions can be connected by switch moves. It is not hard to design concrete examples of graphs for which this is not true with $\varepsilon = 0$. It is known that for $\varepsilon = \infty$, the state space is connected under the switch move when $G$ is biconnected; furthermore, that deciding whether two partitions can be connected by switch moves is PSPACE-complete even when $G$ is planar [2].

The usefulness of the switch move is hampered by the fact that Markov chains built on it tend to mix slowly [23]. As a result, larger reconfiguration moves, that are often more effective on real-world instances, were introduced. The *ReCom move* [13, 14] consists of merging and resplitting two adjacent districts (note that the switch move is a special case of a ReCom move). When designing a Markov chain based on this move, the most common method for resplitting is to draw a random spanning tree of the merged districts and cut an edge such that the resulting connected components form a valid partition. The disadvantage to such a process is that the transition probabilities between partitions appear to be intractable, so that the resulting Markov chain has an unknown stationary distribution. Recently, modifications of the original ReCom Markov chain have been proposed which have computable transition probabilities [4, 6]; however, an accurate description of the stationary distribution still requires the state space to be connected. It is easy to construct a graph $G$ for which the space of valid partitions is not ReCom-connected for $\varepsilon = 0$ (even for a $6\times6$ grid graph [6]). It is known [3] that the state space is connected whenever $G$ is connected and $\varepsilon = \infty$, and also when $G$ is Hamiltonian and $\varepsilon \geq 2$; deciding whether two partitions can be connected by ReCom moves is PSPACE-complete even when $G$ is a triangulation.

**Contributions.** In this paper, we introduce a continuous model for redistricting and ReCom moves, where the districts can be arbitrary connected polygons (with real coordinates) in a polygonal domain (Section 2). While the configuration space in this setting contains infinitely many maps, we prove that it is always connected under ReCom moves. Our proof is constructive, and provides an upper bound on the minimum number of ReCom moves between any two maps in terms of the number of districts $k$ and the complexity of the district maps $n$ (i.e., the number of vertices of all polygons in the initial and target maps). We start with the first nontrivial case, $k = 3$ districts in a unit square domain, and show that between any two maps of complexity $O(n)$, there is a reconfiguration path consisting of $O(\log n)$ ReCom moves (Theorem 9 in Section 3). Importantly, the complexity of the map remains $O(n)$ in all intermediate steps. Our reconfiguration algorithm generalizes to $k$ districts in an arbitrary polygonal domain, using a recursion of depth $O(\log k)$. It yields an $\exp(O(\log k \log\log n))$

bound on the number of ReCom moves between two maps; however, for the complexity of intermediate maps we obtain only a weaker bound of $n^{k^{O(1)}}$ (Theorem 10 in Section 4). On the other hand, we show that (even for $k = 3$) the diameter of the configuration space is infinite by constructing pairs of maps which require arbitrarily large numbers of ReCom moves to connect (Theorem 12 in Section 5). The number of moves for these examples grows logarithmically with the complexity of the maps, thereby providing a lower bound which perfectly matches our upper bound.

## 2 Preliminaries

A *region* is a connected set in $\mathbb{R}^2$ bounded by one or more pairwise disjoint Jordan curves. A *$k$-district map* $M(\mathcal{R}) = \{D_1, \dots, D_k\}$ is a decomposition of a region $\mathcal{R}$ into $k$ interior-disjoint regions (that is, $\mathcal{R} = \bigcup_{i=1}^{k} D_i$ and $\operatorname{int}(D_i) \cap \operatorname{int}(D_j) = \emptyset$ for $i \neq j$), where $\mathcal{R}$ is the *domain*, and $D_1, \dots, D_k$ are the *districts* of the map. We may refer to $M(\mathcal{R})$ simply as $M$ if $\mathcal{R}$ is clear from the context. A *recombination* move (for short, *ReCom*) takes a map $M(\mathcal{R})$ and two districts $D_i, D_j \in M(\mathcal{R})$ and returns a new district map of the same domain $M'(\mathcal{R}) = M(\mathcal{R}) \setminus \{D_i, D_j\} \cup \{D'_i, D'_j\}$. A recombination is *area-preserving* if $\operatorname{area}(D_i) = \operatorname{area}(D'_i)$ and $\operatorname{area}(D_j) = \operatorname{area}(D'_j)$. Two $k$-district maps, $M(\mathcal{R}) = \{D_1, \dots, D_k\}$ and $M'(\mathcal{R}) = \{D'_1, \dots, D'_k\}$, on a domain $\mathcal{R}$ are *area-compatible* if there is a permutation $\pi : \{1, \dots, k\} \to \{1, \dots, k\}$ such that $\operatorname{area}(D_i) = \operatorname{area}(D'_{\pi(i)})$ for all $i = 1, \dots, k$.

We assume that the domain $\mathcal{R}$ is a simple polygon, and each district is a connected polygon (possibly with holes). The *configuration space* of a map $M(\mathcal{R})$ is the set of all polygonal district maps on $\mathcal{R}$ that are area-compatible with $M(\mathcal{R})$. We define the *complexity* of a map $M$ as the total number of vertices on the boundaries of all districts in $M(\mathcal{R})$. We show (in Section 4) that w.l.o.g. we may assume a unit square domain $\mathcal{R} = [0, 1]^2$. The *area* of a polygon $P$, denoted $\operatorname{area}(P)$, is either the Euclidean area of $P$ or the integral $\int_P \delta$ of some nonnegative integrable density function $\delta : \mathcal{R} \to \mathbb{R}_{\geq 0}$. We show (Theorem 10) that between any pair of area-compatible district maps there is a sequence of area-preserving recombinations (i.e., the configuration space of area-compatible district maps is connected).

**Weak Representation.** In intermediate maps of a ReCom sequence, we use infinitesimally narrow *corridors* to keep the districts connected. In order to handle narrow corridors efficiently, we rely on a compressed representation of district maps using *weak embeddings* (defined below), where each corridor is represented by a polygonal path; see Fig. 2. The compressed representation has two key advantages: (1) We may assume that corridors have zero area; and (2) we may reduce the total number of vertices by representing several parallel corridors by overlapping polygonal paths (with shared vertices). In Sections 3–4, we construct a sequence of ReCom moves on compressed maps. We show (in Proposition 1 below) that the polygonal paths can be thickened into narrow corridors in each stage of the ReCom sequence to produce a ReCom sequence in which the districts are simple polygons.

An *embedding* of a planar graph $G$ is an injective function from $G$ (seen as a 1-dimensional topological space) to $\mathbb{R}^2$; intuitively it is a drawing of $G$ in which edges can intersect only at common endpoints. A *weak embedding* of $G$ is a continuous function from $G$ to $\mathbb{R}^2$ such that, for every $\varepsilon > 0$, each vertex can be moved by at most $\varepsilon$ and each edge can be replaced by a curve within Fréchet distance $\varepsilon$ to form an embedding of $G$ (i.e., an *$\varepsilon$-perturbation* of a weak embedding is an embedding). In particular, a *simple polygon* is a piecewise linear embedding of a cycle and the region bound by it; and a *weakly simple polygon* is a piecewise linear weak

embedding of a cycle and the region bounded by it. A *polygon* (with possible holes) is a simple polygon with pairwise disjoint simple polygons (holes) removed. Similarly, a *weak polygon* is a weakly simple polygon with pairwise disjoint weakly simple polygons removed.

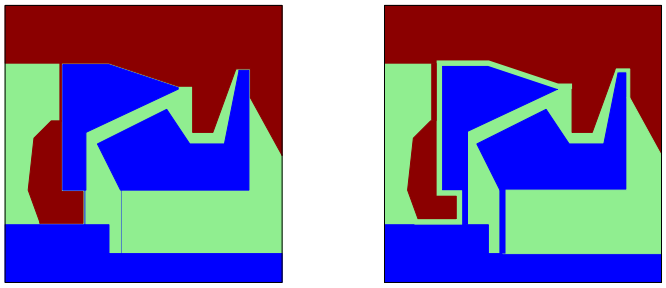


**Figure 2** An example of a weak embedding of a map. Left: multiple corridors connect disconnected regions. Right: the corridors are thickened to create three simple polygons.

For a district map $M$, the boundaries of the districts jointly form a straight-line embedding of some abstract graph $G$. By identifying edges on opposite sides of narrow corridors, we obtain a weak embedding of $G$. In a weak embedding, two or more corridors may overlap, and we maintain a linear order among all overlapping corridors.

We use the machinery introduced by Akitaya et al. [1] (based on earlier work [12, 7]); see also [5, 16]. A weak embedding of $G$ is a piecewise linear map of $\varphi : G \to \mathbb{R}^2$. The *image graph* $H$ is a planar straight-line graph formed by the image $\varphi(G)$, where overlapping vertices (edges) of $G$ are mapped to the same vertex (edge). A *weak representation* of $G$ comprises of a weak embedding $\varphi$ and a linear order of overlapping edges of $\varphi(G)$ along each edge of $H$. We define an *$\varepsilon$-thickening* of $H$ so that $G$ admits an embedding $\psi$ into the $\varepsilon$-thickening of $H$ so that the Fréchet distance between $\varphi$ and $\psi$ is at most $\varepsilon$. We call $\psi$ an *$\varepsilon$-perturbation* of the weak representation if the order of the edges of $G$ in the neighborhood of an edge of $H$ agrees with the given linear order. It is known that if $G$ has $n$ vertices, then an $\varepsilon$-perturbation $\psi(G)$ with $O(n)$ vertices can be computed in $O(n \log n)$ time [1].

**Weak Representation for ReCom Sequences.** We construct a ReCom sequence in two passes: The first pass operates on a generic $\varepsilon$-perturbation, where the *area* of each district is given by the weak representation (hence the corridors have zero area). The second pass then expands the weak representations into an $\varepsilon$-perturbation, using Proposition 1 below (see the full version of the paper for omitted proofs), where each district is a simple polygon with the desired area. Note that the number of moves is determined in the first pass.

▶ **Proposition 1.** *Given two area-compatible $k$-district maps and a sequence of area-preserving ReCom moves where districts in intermediate maps are weak polygons with $O(n)$ vertices, we can compute a sequence of area-preserving ReCom moves of the same length where the districts in intermediate maps are all polygons with $O(n)$ vertices.*

We define the *compressed complexity* of a district map as the number of vertices in the image graph $H$ of the weak representation (that is, repeated vertices are counted only once). The number of ReCom moves produced by our algorithm in Sections 3–4 depends on the compressed complexity. Using $\varepsilon$-perturbations would increase the complexity of maps. For this reason, it is also useful in our analysis to convert an $\varepsilon$-perturbations to a weak embedding which we do by applying the inverse of the operations described here. Throughout this paper we use set operations on weak polygons such as $D_1 \cup D_2$ where $D_1$ and $D_2$ are weak polygons. Let $D_1'$ and $D_2'$ be the polygons obtained by the $\varepsilon$-perturbation defined in Proposition 1. We define $D_1 \cup D_2$ to be the weak polygon obtained from $D_1' \cup D_2'$.

# 3 Reconfiguration for Three Districts

In this section, we consider maps with three districts with a total of $n$ vertices in a unit square domain $\mathcal{R} = [0,1]^2$. We show that any 3-district map $M(\mathcal{R}) = \{D_1, D_2, D_3\}$ can be transformed by a finite sequence of ReCom moves into an area-compatible *canonical map* in which the districts are axis-aligned rectangles, $Q_1$, $Q_2$ and $Q_3$, of unit width such that $\text{area}(Q_i) = \text{area}(D_i)$ for $i = 1, 2, 3$.

## 3.1 Overview of the Algorithm

Our algorithm for transforming a map into the canonical map consists of three stages, each containing multiple ReCom moves:

- *Preprocessing* (Section 3.2). In this stage, we ensure that our three districts are ordered top to bottom in a well-defined way, and the middle district has the largest area. Moves needed: $O(1)$.
- *Gravity moves* (Section 3.3). We perform three ReCom moves to place the districts into their final positions, with the possible exception of corridors. Moves needed: 3.
- *Exchange sequences* (Section 3.5). Corridors maintaining connectivity are carefully removed, using a tree representation to determine a move that simultaneously removes a constant fraction of corridors. Moves needed: $O(\log n)$.

## 3.2 Preprocessing: Ordering Property

First we transform the three given districts into simple polygons if necessary. While there is a district $D_i$ that is a polygon with holes, there is an adjacent district $D_j$ contained within a hole. Recombine $D_i$ and $D_j$ to create a single-edge corridor between $D_j$ and the outer boundary of $D_i$. Next, we create corridors, if necessary, such that each district touches both the left and right sides of $\mathcal{R}$. While there is a district that is not adjacent to the left (resp., right) side $s$ of the $\mathcal{R}$, let $D_i$ be such a district closest to $s$ and let $D_j$ be an adjacent district that already touches $s$; then we recombine $D_i$ and $D_j$ and append to $D_i$ a shortest path to $s$ along the boundary of $D_j$. Thus, both districts remain simply connected. As all corridors run along existing vertices of the three districts, the complexity of the map does not increase. This stage takes $O(1)$ ReCom moves.

After preprocessing, the intersection of each district with the left (resp., right) side of the square domain is connected; and the order of these intersections is the same on both sides, or else two districts would cross. Therefore, the districts can be ordered from top to bottom.

We also need to establish the property that the middle district has the largest area. This can be done trivially with a single ReCom move between the middle district and the largest district of the three. We call these properties combined the *ordering property*:

▶ **Definition 2.** *A three district map $M(\mathcal{R}) = \{D_1, \ldots, D_k\}$ satisfies the* ***ordering property*** *if the intersection of each district with the left (resp., right) side of the square domain is connected, and the middle district, as defined by the resulting order from top to bottom, has area greater than or equal to each other district.*

We assume that the districts are simple polygons in the unit square with a total of $n$ vertices and describe the details of the recombination moves as we use them in the algorithm. To reconfigure the districts into their canonical positions, apart from possible corridors, we perform three gravity moves.

## 3.3 Gravity Move

Assume that $M$ is a 3-district map satisfying the ordering property, with districts labeled $D_1$ (red), $D_2$ (green), and $D_3$ (blue) from top to bottom. We describe the move $\textsc{Gravity}(D_1, D_2)$, which recombines the red and green districts; refer to Figures 3–4. Let $P = D_1 \cup D_2$, which is a weakly simple polygon by the ordering property. By continuity, there exists a horizontal line $\ell$ (that we call the *waterline*) that partitions the plane into upper and lower halfplanes $\ell^+$ and $\ell^-$, resp., such that $\text{area}(P \cap \ell^+) = \text{area}(D_1)$ and $\text{area}(P \cap \ell^-) = \text{area}(D_2)$. We shall define new districts $D_1'$ and $D_2'$, resp., that contain $P \cap \ell^+$ and $P \cap \ell^-$.

Note, however, that $P \cap \ell^+$ and $P \cap \ell^-$ may be disconnected. We then reconnect disjoint components of each district by corridors along the boundary of $P$; see Fig. 4. Note that, by the ordering property, there is a path $\pi$ on the boundary of $D_3$ (blue) between the left and right side of the domain $\mathcal{R}$. If there are two or more components of $P \cap \ell^+$, they are separated by blue and, therefore they all touch the path $\pi$. Therefore $(P \cap \ell^+) \cup \pi$ is a connected region. Similarly, $(P \cap \ell^-) \cup \pi$ is also connected.

We define a *red graph* as follows: the vertices are the connected components of $P \cap \ell^+$ and edges are minimal arcs along $\pi \cap \ell^-$ that connect two distinct components of $P \cap \ell^+$. Since $(P \cap \ell^+) \cup \pi$ is connected, then the red graph is connected. Consider an arbitrary spanning tree of the red graph, and add its edges (as corridors) to the red district along the boundary of $P$. This completes the definition of $D_1'$.

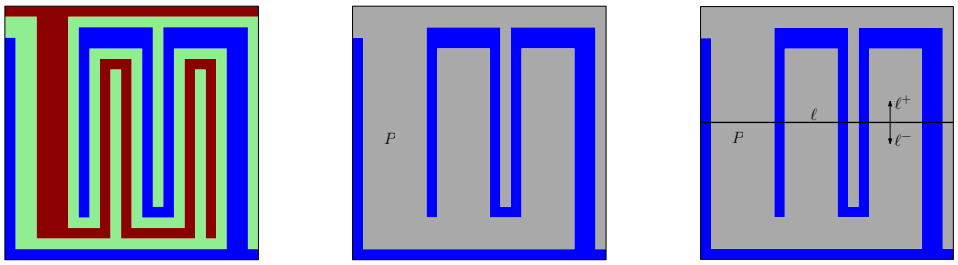



**Figure 3** The setup for a gravity move between the red district $D_1$ and green district $D_2$. Left: a district map satisfying the ordering property. Middle: the union $P = D_1 \cup D_2$ is shown in gray. Right: the horizontal line $\ell$ equipartitions the gray polygon $P$.

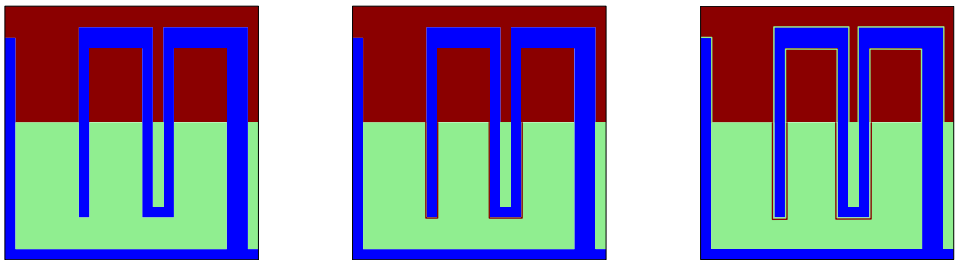


**Figure 4** Constructing the result of a gravity move between red and green on the map in Figure 3. Left: the red region $P \cap \ell^+$ and the green region $P \cap \ell^-$ are each disconnected. Middle: red corridors create a connected red district $D_1'$. Right: green corridors create a connected green district $D_2'$ and restore the ordering property.

Since the blue district is simply connected, each component of $P \cap \ell^-$ also intersects $\ell$ and therefore is adjacent to the red district $D_1'$. Intuitively, we add corridors along $\pi$ "coating" the blue district with green and thus restoring the ordering property. Note that $\pi$ may pass along the boundary of $D_1'$, including all red corridors, and the boundaries of the components

of $P \cap \ell^-$. Formally, we add green corridors at the intersection of $\pi$ and $\partial D_1'$, if such a corridor is parallel to a red corridor, it runs between the blue district and the red corridor. That defines $D_2'$ and concludes the description of the gravity move.

▶ **Lemma 3.** *Assume $D_1$ and $D_2$ are the top two districts on a map satisfying the ordering property. Then* Gravity$(D_1, D_2)$ *is an area-preserving ReCom move that maintains the ordering property.*

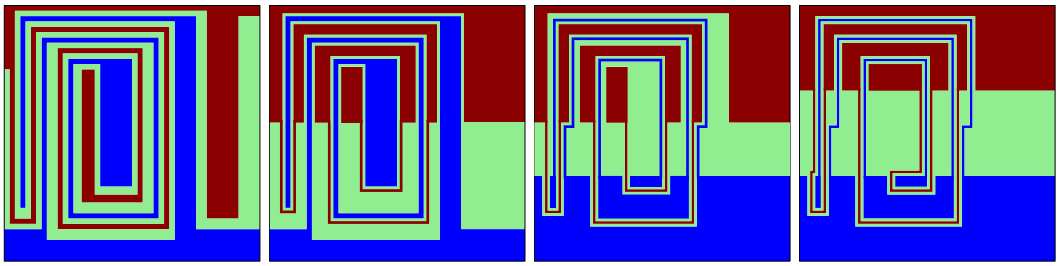
Since each waterline intersects an edge of a district at most once we have:

▶ **Lemma 4.** *Assume $D_1$, $D_2$ and $D_3$ each have at most $m$ vertices. Then* Gravity$(D_1,D_2)$ *produces districts $D_1'$ and $D_2'$, each with at most $O(m)$ vertices.*

The move Gravity$(D_3, D_2)$ is defined analogously: a reflection in a horizontal line that reverses the order of the three districts, such that $D_1' = D_3$ becomes the top district and $D_2' = D_2$ is the middle district, then apply Gravity$(D_1', D_2')$, followed by another reflection.

▶ **Lemma 5.** *Let $M$ be a map satisfying the ordering property, with districts $D_1$, $D_2$ and $D_3$ from top to bottom. Then* Gravity$(D_1, D_2)$ *returns a map that satisfies the ordering property and $D_1'$ is disjoint from $Q_3$ with the possible exception of corridors, where $Q_3$ is the axis-aligned rectangle of the blue district in the canonical map.*

**Proof.** It suffices to show that the horizontal line $\ell$ is above $Q_3$. Lemma 3 yields the rest. By definition, the area below $\ell$ is at least area$(D_2) \geq$ area$(Q_3)$, since $D_2$ has the maximum area of the three districts. Thus, the line $\ell$ is above $Q_3$. ◀



(a) (b) (c) (d)

**Figure 5** An example of a sequence of three gravity moves. (a) A starting configuration; (b) the result of Gravity$(D_1, D_2)$; (c): the result of Gravity$(D_3, D_2)$; (d) the result of the third gravity move Gravity$(D_1, D_2)$.

After three gravity moves each district has positive area only in the regions in their corresponding districts in the canonical configuration (see Figure 5 for an example).

▶ **Lemma 6.** *Let $M$ be a map satisfying the ordering property, with districts $D_1$, $D_2$ and $D_3$ from top to bottom. Then the sequence of three moves* Gravity$(D_1, D_2)$, Gravity$(D_2, D_3)$, *and* Gravity$(D_1, D_2)$ *return a map $M'$ where $D_1$, $D_2$, and $D_3$ are each contained in their canonical rectangles, with the possible exception of some corridors.*

## 3.4 Tree Representation of a Region

After prepocessing and the three gravity moves in Lemma 6, we want to eliminate corridors. We encode the topology of the region $P = D_1 \cup D_2$ in a graph that we use for the Exchange sequence, described below.

We define the *corridor graph* $T(R)$ of a weakly simple polygon $R \subset \mathcal{R}$. A weakly simple polygon has a natural decomposition into pairwise disjoint simple polygons and corridors (polygonal paths). The nodes of $T(R)$ are simple polygons in the decomposition of $R$, and the edges represent corridors between two polygons in $R$. Denote the set of edges by $E(T(R))$. At each node, the rotation of the incident edges represents the counterclockwise order of corridors along the corresponding polygon in $R$. The weight of each node is the area of the corresponding polygon. As corridors have zero thickness, the total weight of the nodes is $W = \text{area}(R)$.

In particular, we want to consider the corridor graph of $P = D_1 \cup D_2$. Assume that $M$ is a 3-district map returned by the three gravity moves in Lemma 6. By the ordering property, we know that the intersection of $D_1$ and $D_2$ is a simple path - either from one side of the square to another or, if $D_1$ is contained in $D_2$, then it is a closed curve. Thus, $P$ is a weakly simple polygon. Let $Q_{12}$ be the union of the two axis-aligned rectangles that contain $D_1$ and $D_2$ in the canonical configuration. Then, the nodes of $T(P)$ are simple polygons in $P \cap Q_{12}$ (regions bounded by corridors of $D_3$) and the edges are corridors in $R \setminus Q_{12}$ that connect two such polygons (corridors of $D_1$ and $D_2$ running through $Q_3$). Note, however, that a corridor in $P$ may be the union of three parallel corridors in $D_2$, $D_1$, and $D_2$, resp.; see Fig. 6. Since $P$ is a weakly simple polygon, $T(P)$ is a tree; see Fig. 6. Note that the number of vertices in $T(P)$ is bounded above by the compressed complexity of the map and that many different maps can have the same corridor graph.

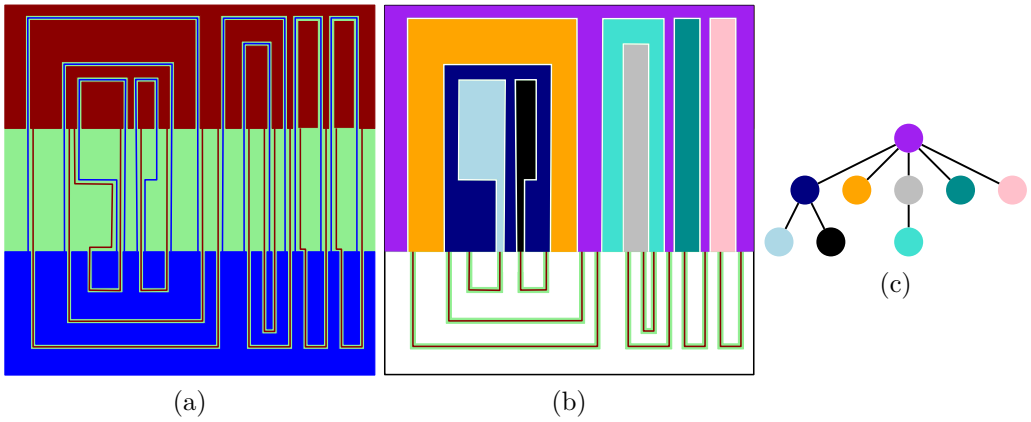



**Figure 6** (a) A map $M$ after 3 Gravity moves. (b) The nodes of the corridor graph $T(P)$ correspond to connected components of $P \cap Q_{12}$, indicated by distinct colors. (c) The corridor graph $T(P)$ encodes the topology of $P$.

We use the corridor graph $T(P)$ to eliminate corridors. Consider what happens if the tree has a leaf that is entirely part of the green district (see Fig. 7). This means that by doing a gravity move between green and blue we can eliminate the green and blue corridors adjacent to this leaf, removing the leaf from the tree altogether. Our goal is therefore to create a part of the tree which is entirely green.

The *centroid* of a vertex-weighted tree of total weight $W$ is a vertex whose removal partitions the tree into subtrees of weight at most $\frac{W}{2}$ each. Jordan [19] proved that every tree (with unit weights) has a centroid; this was perhaps the oldest separator theorem [20, 22]. The result extends to weighted trees: a greedy algorithm finds the centroid in linear time.

Let $c$ be a centroid of $T(P)$, and assume that $T(P)$ is rooted at $c$. A subtree of $T(P)$ is *contiguous* if it consists of the centroid $c$, some children of $c$ that are consecutive in the rotation order of $c$, and all their descendants in $T(P)$.

▶ **Lemma 7.** *There exists a contiguous subtree $T^*$ of $T(P)$ such that:* (i) *$T^*$ contains at least $\frac{1}{3}$ of the vertices of $T(P)$, and* (ii) *the weight of $T^*$ is at most $\frac{W}{2} + w(c)$, where $w(c)$ is the weight of the centroid $c$.*

**Proof.** By the definition of the centroid $c$, the removal of $c$ produces a forest $T(P) - c$, where the weight of each component (tree) is at most $\frac{W}{2}$. Partition these $\deg(c)$ trees into up to three forests of consecutive subtrees such that each forest has weight at most $\frac{W}{2}$ as follows. Begin with a partition into $\deg(c)$ forests, each containing a single tree, and maintain their cyclic order around $c$. While there are two consecutive forests whose combined weight is at most $\frac{W}{2}$, merge them into a single forest. The while loop terminates with three or fewer forests: Indeed, for four or more forests, the combined weight of at least one of the consecutive pairs would be at most $\frac{W}{2}$ by the pigeonhole principle. Since we partition $T(P) - c$ into three forests, one of them contains at least $\frac{1}{3}$ of the vertices $T(P) - c$. Adding $c$ to this forest, we obtain a contiguous subtree of $T(P)$ containing at least $\frac{1}{3}$ of the vertices of $T(P)$. ◀

## 3.5 Exchange Sequence

We now describe the *exchange sequence*, a sequence of three ReCom moves, which eliminates a fraction of the corridors and reduces the (compressed) complexity of the map. Assume we are given a 3-district map $M$ satisfying the ordering property. As before, label its districts red, green, and blue from top to bottom. We further require that there exist two horizontal lines $\ell_1$ and $\ell_2$ such that red has positive area only above $\ell_2$, blue has positive area only below $\ell_1$, and green has positive area only between $\ell_1$ and $\ell_2$ (cf. Lemma 6). See Figure 7 for an example.

Let $c$ be a centroid of $T(P)$, where $P = D_1 \cup D_2$ and let $T^*$ be a contiguous subtree of $T(P)$ rooted at the centroid, as in Lemma 7. The exchange sequence consists of the following three ReCom moves:

1. ReCom green and red: Let $Q$ denote the regions of $T^*$ except for the region corresponding to node $c$. First make $Q$ green. Then partition the remaining region $P \setminus Q$ with a gravity-like move as follows. Apply a GRAVITY move w.r.t. $P \setminus Q$ to subdivide it into two weak polygons of areas $\text{area}(D_1)$ for red and $\text{area}(D_2) - \text{area}(Q)$ for green; see Fig. 7. After this ReCom move, $D_1$ is weakly simple and $D_2$ is a weak polygon (in which $D_1$ is a hole if $Q$ is a weak polygon with a hole).
2. ReCom green and blue removing unnecessary green and blue corridors simultaneously as follows. Remove any green and blue monochromatic corridors corresponding to all edges of $T^*$. Note that this merges some nodes of $T(D_3)$ (see Fig. 7), and creates cycles in $T(D_3)$. While there is a cycle in $T(D_3)$ remove a blue corridor in an edge of $T(D_3)$ in a cycle. As this process modifies only green and red, it requires a single ReCom move. After this ReCom move, $D_3$ is a weakly simple polygon and $D_2$ is a weakly simple or weak polygon.
3. ReCom green and red with a GRAVITY move, restoring the ordering property.

▶ **Lemma 8.** *Let $M = \{D_1, D_2, D_3\}$ be a 3-district map with the ordering property, and $M' = \{D_1', D_2', D_3'\}$ the map returned by an EXCHANGE sequence on $M$. Let $P = D_1 \cup D_2$ and $P' = D_1' \cup D_2'$. Then, $M'$ satisfies the ordering property, and $|E(T(P'))| \leq \frac{2}{3}|E(T(P))|$.*

## 3.6 Full Reconfiguration Algorithm

Overall, the algorithm for a 3-district map $M([0,1]^2) = \{D_1, D_2, D_3\}$ works as follows: after a preprocessing phase of $O(1)$ ReCom moves, apply the sequence of three moves GRAVITY$(D_1, D_2)$, GRAVITY$(D_2, D_3)$, and GRAVITY$(D_1, D_2)$; compute the corridor graph

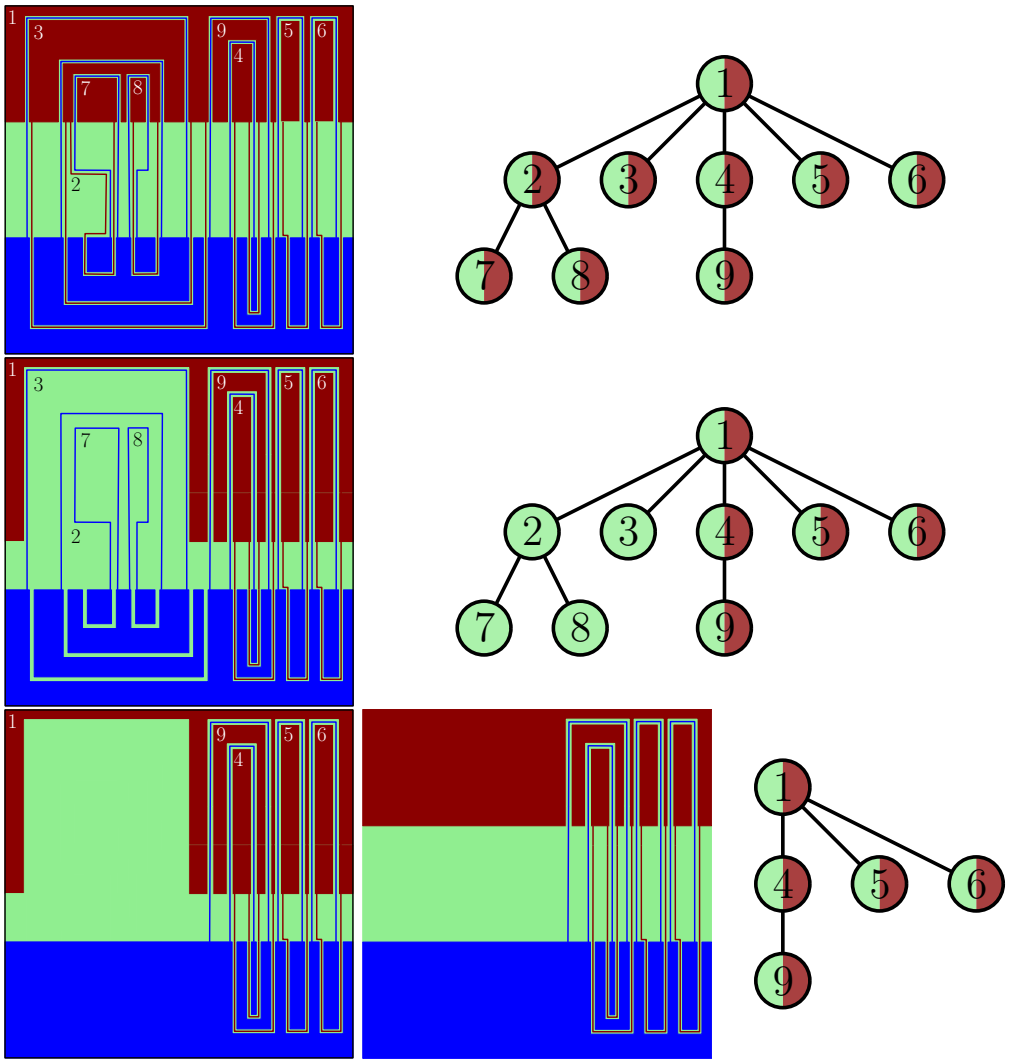



**Figure 7** An exchange sequence, shown with maps (left) and corresponding tree representations (right). Top: a map returned by a sequence of three gravity moves. Middle: using node 1 as the centroid $c$ and filling the subtree containing nodes 2, 3, 7 and 8 with green. Bottom: removing unnecessary corridors and performing a gravity move.

$T(P)$ for $P = D_1 \cup D_2$; while $T(P)$ has two or more nodes, apply an exchange sequence. Once $T(P)$ has one node, $\textsc{Gravity}(D_1, D_2)$ yields the canonical configuration.

▶ **Theorem 9.** *Given a 3-district map $M([0,1]^2) = \{D_1, D_2, D_3\}$ of complexity $n$, there is a sequence of $O(\log n)$ ReCom moves that transforms it into a canonical map. Furthermore, the districts in each intermediate map are polygons with $O(n)$ vertices and at most one hole.*

**Proof.** After preprocessing, three Gravity moves bring the districts into canonical form with the possible exception of corridors. Each exchange sequence eliminates a constant fraction of corridors by Lemma 8. After $O(\log n)$ ReCom moves we then obtain the canonical configuration.

The algorithm described above produces a sequence of ReCom moves, where the districts in intermediate maps are weak polygons. By Proposition 1, these maps can successively be perturbed into polygons. This completes the proof of the first claim.

It remains to show that the districts in each intermediate map are polygons with $O(n)$ vertices and at most one hole. By construction, the only possible hole appears in the green district after the first step of the EXCHANGE sequence. Each of the $O(1)$ ReCom moves in preprocessing adds a corridor with $O(n)$ vertices, and so each district has $O(n)$ vertices at the end of preprocessing. By Lemma 4, each gravity move increases the number of vertices by a constant factor. After three gravity moves, each district still has $O(n)$ vertices.

The algorithm applies $O(\log n)$ exchange sequences. At the end of every exchange sequence, the districts are in canonical form with the exception of corridors. Each exchange sequence removes some of the corridors, and does not create new corridors. It should be clear that the complexity of the blue district only decreases since corridors are only eliminated and never created. Note that intermediate ReCom moves within an EXCHANGE sequence (step 1) may add $O(n)$ new vertices to the red district. In an exchange sequence, the 1st ReCom move is a GRAVITY move w.r.t. a sub-polygon, and creates only $O(n)$ new vertices by Lemma 4. The 2nd ReCom move eliminates corridors (and the corresponding vertices); and the 3rd ReCom move eliminates any other vertices created in the 1st move of the sequence. Thus, the complexities of the red and green districts decrease after one EXCHANGE sequence.

Finally, when we perturb all weak polygons into polygons in the entire ReCom sequence, the number of vertices remains $O(n)$ for each district by Proposition 1. ◀

## 4 Reconfiguration for $k$ Districts

We generalize our algorithm to an arbitrary number of districts, using recursion. For any $3 \le k \le n$, an *instance* $I = (M(\mathcal{R}), M'(\mathcal{R}), \delta)$ of the problem consists of two area-compatible $k$-district maps $M(\mathcal{R}) = \{D_1, \ldots, D_k\}$ and $M'(\mathcal{R}) = \{D'_1, \ldots, D'_k\}$, where $\mathcal{R}$ is a weak polygon with at most one hole, and $\delta$ is a density function. We define the complexity of $I$ (denoted $|I|$) as the pair $(k, n)$, where $n$ is the maximum over the compressed complexities of $M$ and $M'$, and the complexities of all districts $D_i$ and $D'_i$ ($i \in \{1, \ldots, k\}$). The overall recursive strategy goes as follows (see the full paper for the details): First construct a piecewise linear retraction from a (possibly punctured) unit square $\mathcal{S}$ to $\mathcal{R}$, and extend $M$ and $M'$ to two maps on $\mathcal{S}$. If $k \ge 4$, then group the $k$ districts into three *superdistricts*, each containing $\lfloor k/3 \rfloor$ or $\lceil k/3 \rceil$ districts; and run the algorithm in Section 3 on the superdistricts. Note that each ReCom move on a pair of superdistricts is an instance of our problem with fewer districts, which can be solved recursively. The retraction then transforms the ReCom sequence on $\mathcal{S}$ to a ReCom sequence on $\mathcal{R}$. We analyze the recursion to give a bound on the number of ReCom moves.

▶ **Theorem 10.** *Given any two area-compatible polygonal $k$-district maps of complexity at most $n$ in a simply connected domain,* $\exp(O(\log k \log\log n)) = (\log n)^{O(\log k)} = k^{O(\log\log n)}$ *ReCom moves are sufficient to transform one into the other. Furthermore, the complexity of each map in intermediate steps is* $n^{k^{O(1)}}$.

**Proof.** For $3 \le k \le n$, let $T(k, n)$ denote the minimum number of ReCom moves that can transform any polygonal $k$-district map to any other with compatible areas, and the domain as well as each district is a polygon with at most $n$ vertices. From an instance $I(k, n)$, our algorithm makes $O(\log n)$ recursive calls of the form $I(\frac{2k}{3}, c \cdot n)$, where $c$ is a constant. Then,

$$T(k, n) \le O\left(T\left(\frac{2k}{3}, c \cdot n\right) \cdot \log n\right) + O(k).$$

The height of the recursion tree is $O(\log k)$ and the maximum branching factor is $O(\log(n \cdot c^{\log k})) = O(\log n + \log k) = O(\log n)$ since $k < n$. Then $T(k, n)$ solves to $\exp(O(\log k \log\log n)) = (\log n)^{O(\log k)} = k^{O(\log\log n)}$. By Proposition 1, we can convert the ReCom sequence on weak representation to a ReCom sequence of the same length in which all districts are simple polygons.

The analysis above prioritized the number of ReCom moves, rather than the complexity of the map at intermediate steps. For instance, consider the recursion that simulates a ReCom move of superdistricts transforming a $k$-district map $M(\mathcal{R})$ into $M'(\mathcal{R})$. Our algorithm recurses on a $\frac{2k}{3}$-district map of complexity $c \cdot n$ on a punctured square $\mathcal{S}$, which yields a sequence of $O(\log n)$ ReCom moves. However, to convert this into a sequence of ReCom moves on $k$-district maps, one must apply a retraction $\mathcal{H}^*$ (in the full paper) to every intermediate map, retracting a weakly simple polygon $H$ to its boundary $\partial H$. Since the complexity of $H$ could be $\Omega(n)$, $H$ might cross the same district $\Omega(n)$ times, which causes $\mathcal{H}^*$ to push the district into $\Omega(n)$ narrow corridors along the boundary of $H$. This might cause the complexity of the district to increase to $\Omega(n^2)$ in intermediate steps. The retraction $\mathcal{H}^*$, described in the full paper, ensures that the complexity goes up from $n$ to at most $O(n^2)$ after applying $\mathcal{H}^*$ in each recursive step. Since the depth of the recursion tree is $O(\log k)$, the maximum complexity of all intermediate maps is $n^{2^{O(\log k)}} = n^{k^{O(1)}}$. Note that this does not increase the number of ReCom moves since $M$ and $M'$ are determined in the parent level, and $\mathcal{H}^*$ is only applied to recover intermediate steps between $M$ and $M'$, which are obtained from lower complexity maps in the children level. ◀

## 5 Lower Bound Construction

This section shows that $\Omega(\log n)$ ReCom moves are sometimes necessary to transform a given map of complexity $n$ into canonical form, even for three districts of equal areas in $[0, 1]^2$.

**Overview.** We describe an initial map with 3 districts in a unit square, and show that after $k$ ReCom moves, each district contains an arc of a specific combinatorial pattern (defined below). These arcs are defined recursively, each iteration roughly tripling the complexity of the arcs. Thus the total complexity of the arcs in iteration $\ell$ is $O(3^\ell)$. The initial district map is a thickening of one of these arcs after $m \geq 6$ iterations. We show that if each district contains an arc from iteration $\ell$, then after a recombination they each contain an arc of iteration $\ell - 4$. In the canonical configuration, each district can only contain arcs of iteration 1. Then, the number of recombinations from the initial district map to the canonical configuration is at least linear in the number of iterations.

**Construction.** We first describe the family of simple arcs $F_\ell$, for all $\ell \in \mathbb{N}_0$, mentioned in the overview. All arcs in $F_\ell$ will start at the $\varepsilon$-neighborhood of the left side of the square and end at the $\varepsilon$-neighborhood of the right side, crossing the middle section $3^\ell$ times. Each family $F_\ell$ can be described with a combinatorial pattern, namely, the order in which the arcs traverse the $3^\ell$ segments in the middle section of the square. In the base case, $F_0$ is the set of arcs that cross the middle section only once. Given an arc $\gamma_\ell \in F_\ell$, we describe an arc $\gamma_{\ell+1} \in F_{\ell+1}$. We construct two arcs, $\gamma_\ell^+, \gamma_\ell^- \in F_\ell$, that closely follow $\gamma_\ell$ on the left and on the right, respectively, and are mutually noncrossing. Then $\gamma_{\ell+1}$ is the concatenation of $\gamma_\ell$, the reverse of $\gamma_\ell^-$, and $\gamma_\ell^+$, where two consecutive arcs are connected by short arcs in the left and right $\varepsilon$-neighborhoods of the square; see Fig 8. Let $F_{\ell+1}$ be the family of all arcs with the same combinatorial pattern as $\gamma_{\ell+1}$. The following observation follows by construction.

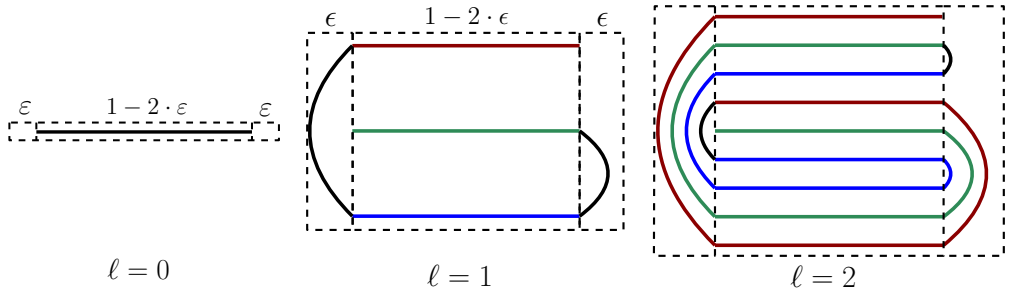



**Figure 8** The first three levels of the recursive construction for arcs in $F_\ell$, for $\ell \in \{0,1,2\}$. Note that the blue, green, and red arcs for $\ell = 2$ each resemble a copy of the entire stage for $\ell = 1$.

▶ **Observation 11.** *For $0 \le j \le \ell$, we can partition every arc $\gamma_\ell \in F_\ell$ into $3^j$ arcs in $F_{\ell-j}$.*

**Initial Map.** The initial map is drawn relative to an arc $\gamma_m \in F_m$ whose middle segments are equally spaced horizontal line segments in the unit square. The map is a "thickening" of $\gamma_m$ where the middle section is partitioned into $3^m$ rectangles of equal area. Each of the three districts is created based on one of three rough copies of $\gamma_{m-1}$, i.e., the $(m-1)$-anchors of $\gamma_m$. We use the portions of the anchors of $\gamma_m$ in the $\varepsilon$-neighborhoods of the vertical sides of the unit square to construct corridors that make each district connected.

In the full paper, we show that any ReCom move can only make constant progress (in the number of iterations) towards the canonical map.

▶ **Theorem 12.** *There exist two area-compatible 3-district maps, $M$ and $M'$, both with complexity $O(n)$, such that $\Omega(\log n)$ ReCom moves are necessary to reconfigure $M$ into $M'$, even when the districts in both maps are axis-aligned orthogonal polygons with vertices on an integer grid of size $O(n) \times O(n)$.*

## 6 Conclusions

We have shown that (in our continuous setting) any pair of area-compatible district maps can be reconfigured into each other by a sequence of area-preserving recombination moves. Though the discrete version of this result remains unsolved (see Related Work), our result suggests that for any two maps, with a discretization of the geographic domain which is granular enough, we can connect them by ReCom moves. However, establishing quantitative bounds on the necessary granularity is left for future work.

Between 3-district maps, the number of recombination moves is $O(\log n)$, where $n$ is the combinatorial complexity of the maps, matching our worst-case lower bound of $\Omega(\log n)$. Between $k$-district maps, for $k \ge 4$, we construct a sequence of $\exp(O(\log k \log\log n)) = (\log n)^{O(\log k)}$ ReCom moves. It remains an open problem whether the number of moves can be reduced to be polynomial in both $k$ and $n$. For $k \ge 4$ districts, our algorithm uses a recursion of depth $O(\log k)$. However, this approach increases the complexity of intermediate maps to $n^{k^{O(1)}}$. It is also an open problem whether there exists a sequence of ReCom moves where the complexity of intermediate maps remains polynomial in both $k$ and $n$.

## References


1 Hugo A Akitaya, Greg Aloupis, Jeff Erickson, and Csaba D Tóth. Recognizing weakly simple polygons. *Discrete & Computational Geometry*, 58(4):785–821, 2017.

2 Hugo A. Akitaya, Matthew D. Jones, Matias Korman, Oliver Korten, Christopher Meierfrankenfeld, Michael J. Munje, Diane L. Souvaine, Michael Thramann, and Csaba D. Tóth. Reconfiguration of connected graph partitions. *Journal of Graph Theory*, 102(1):35–66, 2023. doi:10.1002/jgt.22856.

3 Hugo A. Akitaya, Matias Korman, Oliver Korten, Diane L. Souvaine, and Csaba D. Tóth. Reconfiguration of connected graph partitions via recombination. *Theoretical Computer Science*, 923:13–26, 2022. doi:10.1016/j.tcs.2022.04.049.

4 Eric A. Autry, Daniel Carter, Gregory Herschlag, Zach Hunter, and Jonathan C. Mattingly. Multi-scale merge-split Markov chain Monte Carlo for redistricting, 2020. arXiv:2008.08054.

5 Thomas Bläsius, Simon D. Fink, and Ignaz Rutter. Synchronized planarity with applications to constrained planarity problems. In *Proc. 29th European Symposium on Algorithms (ESA)*, volume 204 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl, 2021. doi:10.4230/LIPIcs.ESA.2021.19.

6 Sarah Cannon, Moon Duchin, Dana Randall, and Parker Rule. Spanning tree methods for sampling graph partitions, 2022. arXiv:2210.01401.

7 Hsien-Chih Chang, Jeff Erickson, and Chao Xu. Detecting weakly simple polygons. In *Proc. 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1655–1670, 2014. doi:10.1137/1.9781611973730.110.

8 Jowei Chen and Jonathan Rodden. Unintentional gerrymandering: Political geography and electoral bias in legislatures. *Quarterly Journal of Political Science*, 8(3):239–269, 2013. doi:10.1561/100.00012033.

9 Jowei Chen and Jonathan Rodden. Cutting through the thicket: Redistricting simulations and the detection of partisan gerrymanders. *Election Law Journal*, 14(4):331–345, 2015. doi:10.1089/elj.2015.0317.

10 Maria Chikina, Alan Frieze, and Wesley Pegden. Assessing significance in a Markov chain without mixing. *Proceedings of the National Academy of Sciences*, 114(11):2860–2864, 2017. doi:10.1073/pnas.16175401.

11 Vincent Cohen-Addad, Philip N Klein, and Neal E Young. Balanced centroidal power diagrams for redistricting. In *Proc. 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 389–396, 2018.

12 Pier Francesco Cortese, Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. On embedding a cycle in a plane graph. *Discrete Mathematics*, 309(7):1856–1869, 2009. doi:10.1016/j.disc.2007.12.090.

13 Daryl DeFord and Moon Duchin. Redistricting reform in Virginia: Districting criteria in context. *Virginia Policy Review*, 12(2):120–146, 2019.

14 Daryl DeFord, Moon Duchin, and Justin Solomon. Recombination: A family of Markov chains for redistricting. *Harvard Data Science Review*, March 2021. doi:10.1162/99608f92.eb30390f.

15 Benjamin Fifield, Michael Higgins, Kosuke Imai, and Alexander Tarr. Automated redistricting simulation using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 29(4):715–728, 2020. doi:10.1080/10618600.2020.1739532.

16 Radoslav Fulek and Csaba D. Tóth. Atomic embeddability, clustered planarity, and thickenability. *J. ACM*, 69(2):13:1–13:34, 2022. doi:10.1145/3502264.

17 Gregory Herschlag, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier, and Jonathan C. Mattingly. Quantifying gerrymandering in north carolina. *Statistics and Public Policy*, 7(1):30–38, 2020. doi:10.1080/2330443X.2020.1796400.

18 Gregory Herschlag, Robert Ravier, and Jonathan C. Mattingly. Evaluating partisan gerrymandering in Wisconsin, 2017. arXiv:1709.01596.

19 Camille Jordan. Sur les assemblages des lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.

20 Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. doi:10.1137/0136016.

21 Jonathan C. Mattingly and Christy Vaughn. Redistricting and the will of the people, 2014. arXiv:1410.8796.

22 Gary L. Miller, Shang-Hua Teng, William P. Thurston, and Stephen A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997. doi:10.1145/256292.256294.

23 Elle Najt, Daryl DeFord, and Justin Solomon. Complexity and geometry of sampling connected graph partitions, 2019. arXiv:1908.08881.

24 Patti B. Saris, Gary King, Jamal Greene, and Moon Duchin. Redistricting and representation. *Bulletin of the American Academy of Arts and Sciences*, 71(2):47–58, 2018.

# Faster Detours in Undirected Graphs

**Shyan Akmal**
MIT EECS and CSAIL, Cambridge, MA, USA

**Virginia Vassilevska Williams**
MIT EECS and CSAIL, Cambridge, MA, USA

**Ryan Williams**
MIT EECS and CSAIL, Cambridge, MA, USA

**Zixuan Xu**
MIT Mathematics, Cambridge, MA, USA

## Abstract

The $k$-Detour problem is a basic path-finding problem: given a graph $G$ on $n$ vertices, with specified nodes $s$ and $t$, and a positive integer $k$, the goal is to determine if $G$ has an $st$-path of length exactly $\mathsf{dist}(s,t)+k$, where $\mathsf{dist}(s,t)$ is the length of a shortest path from $s$ to $t$. The $k$-Detour problem is NP-hard when $k$ is part of the input, so researchers have sought efficient parameterized algorithms for this task, running in $f(k)\,\mathrm{poly}(n)$ time, for $f(\cdot)$ as slow-growing as possible.

We present faster algorithms for $k$-Detour in undirected graphs, running in $1.853^k\,\mathrm{poly}(n)$ randomized and $4.082^k\,\mathrm{poly}(n)$ deterministic time. The previous fastest algorithms for this problem took $2.746^k\,\mathrm{poly}(n)$ randomized and $6.523^k\,\mathrm{poly}(n)$ deterministic time [Bezáková-Curticapean-Dell-Fomin, ICALP 2017]. Our algorithms use the fact that detecting a path of a given length in an undirected graph is easier if we are promised that the path belongs to what we call a "bipartitioned" subgraph, where the nodes are split into two parts and the path must satisfy constraints on those parts. Previously, this idea was used to obtain the fastest known algorithm for finding paths of length $k$ in undirected graphs [Björklund-Husfeldt-Kaski-Koivisto, JCSS 2017], intuitively by looking for paths of length $k$ in randomly bipartitioned subgraphs. Our algorithms for $k$-Detour stem from a new application of this idea, which does not involve choosing the bipartitioned subgraphs randomly.

Our work has direct implications for the $k$-Longest Detour problem, another related path-finding problem. In this problem, we are given the same input as in $k$-Detour, but are now tasked with determining if $G$ has an $st$-path of length *at least* $\mathsf{dist}(s,t)+k$. Our results for $k$-Detour imply that we can solve $k$-Longest Detour in $3.432^k\,\mathrm{poly}(n)$ randomized and $16.661^k\,\mathrm{poly}(n)$ deterministic time. The previous fastest algorithms for this problem took $7.539^k\,\mathrm{poly}(n)$ randomized and $42.549^k\,\mathrm{poly}(n)$ deterministic time [Fomin et al., STACS 2022].

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** path finding, detours, parameterized complexity, exact algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.7

**Related Version** *Full Version*: https://arxiv.org/abs/2307.01781 [1]

**Funding** *Shyan Akmal*: Supported in part by NSF grant CCF-2127597.
*Virginia Vassilevska Williams*: Supported by NSF CCF-2129139.
*Ryan Williams*: Supported by NSF CCF-1909429.

**Acknowledgements** The first author thanks Nicole Wein for several helpful discussions, and Fedor V. Fomin for answering a question about previous algorithms for $k$-Longest Path.





## 1 Introduction

The $k$-Path problem is a well-studied task in computer science:

> $k$-Path
> **Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
> **Determine:** Does $G$ contain a simple path of length $k$ from $s$ to $t$?

For graphs $G$ with $n$ nodes, this problem can be easily solved in $O(kn^k)$ time by enumerating all sequences of $k$ vertices. In the 1980s, Monien [13] showed that the $k$-Path problem is actually fixed-parameter tractable (FPT) in $k$, presenting a $k!\,\mathrm{poly}(n)$ time algorithm solving $k$-Path. Since then, significant research has gone into obtaining faster algorithms for $k$-Path, with better dependence on $k$ (see [3, Table 1] for an overview of the many results in this line of work). This research culminated in the work of Koutis and Williams [11, 16, 12], who showed that $k$-Path can be solved in $2^k\,\mathrm{poly}(n)$ (randomized) time, and Björklund, Husfeldt, Kaski, and Koivisto [3, Section 2], who proved that in undirected graphs, $k$-Path can be solved even faster in $1.657^k\,\mathrm{poly}(n)$ (randomized) time. *Throughout this paper, we assume that algorithms are randomized (and return correct answers with high probability in the stated time bounds), unless otherwise specified.*

The $k$-Path problem is a parameterized version of the NP-complete Longest Path problem, but it is not the only natural parameterization. Various other parameterizations of $k$-Path have been proposed and studied, which we consider in the present paper.

1. **Finding a path of length at least $k$.** Instead of looking for a path of length *exactly* $k$ from $s$ to $t$, one can try to determine the existence of an $st$-path of length *at least* $k$:

   > $k$-Longest Path
   > **Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
   > **Determine:** Does $G$ contain a simple path of length ***at least*** $k$ from $s$ to $t$?

   Observe that in the $k$-Longest Path problem, the length of a solution path is not necessarily bounded as a function of $k$. However, it is known that $k$-Longest Path is also FPT: work of Zehavi [17] and Fomin, Lokshtanov, Panolan, and Saurabh [9] implies that $k$-Longest Path can be solved in $4^k\,\mathrm{poly}(n)$ time. More recently, Eiben, Koana, and Wahlström [7, Section 6.3] proved that over undirected graphs, $k$-Longest Path can be solved in $1.657^k\,\mathrm{poly}(n)$ time, matching the fastest known runtime for $k$-Path.

2. **Finding an $st$-path longer than a polynomial-time guarantee.** Another parameterization for $k$-Path is motivated by the fact that one can find a *shortest path* from $s$ to $t$ in polynomial time. If the shortest path distance $\mathsf{dist}(s,t)$ happens to already be long, then it is actually "easy" to find a long path from $s$ to $t$. Therefore, it is natural to consider the parameterized complexity of searching for an $st$-path that is $k$ edges *longer* than the shortest path length from $s$ to $t$. Our work focuses on these so-called "detour" variants of the path detection problems discussed above.

   > $k$-Detour (a.k.a. $k$-Exact Detour)
   > **Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
   > **Determine:** Does $G$ contain a simple path of length $\mathsf{dist}(s,t)+k$ from $s$ to $t$?

Since $k$-Path efficiently reduces to solving a single instance of $(k-1)$-Detour,[1] the $k$-Detour problem is at least as hard as the classical $k$-Path problem.
The $k$-Detour problem was introduced by Bezáková, Curticapean, Dell, and Fomin [2], who showed that it can be solved by calling polynomially many instances of $\ell$-Path, for path lengths $\ell \leq 2k+1$. Employing the fastest known $k$-Path algorithms, this implies that $k$-Detour can be solved in $2^{2k}\operatorname{poly}(n) = 4^k \operatorname{poly}(n)$ time in general, and even faster over undirected graphs in $1.657^{2k}\operatorname{poly}(n) \leq 2.746^k \operatorname{poly}(n)$ time.

The two parameterizations above can be combined to produce the following problem:

$k$-Longest Detour
**Given:** $k \in \mathbb{N}^+$, a graph $G$, nodes $s$ and $t$.
**Determine:** Does $G$ contain a simple path of length ***at least*** $\mathsf{dist}(s,t)+k$ from $s$ to $t$?

Observe that $k$-Longest Detour is at least as hard as $k$-Longest Path. Unlike the problems discussed above, $k$-Longest Detour over directed graphs is not known to be FPT: in fact, it remains open whether $k$-Longest Detour is in P even for the special case of $k = 1$! However, in undirected graphs, Fomin, Golovach, Lochet, Sagunov, Simonov, and Saurabh [8] showed that $k$-Longest Detour can be reduced to solving $p$-Detour for $p \leq 2k$, and then solving polynomially many instances of $\ell$-Longest Path, for $\ell \leq 3k/2$. Employing the fastest known algorithms for $k$-Detour and $k$-Longest Path as subroutines, this implies that $k$-Longest Detour can be solved over undirected graphs in $7.539^k \operatorname{poly}(n)$ time.

The algorithms for $k$-Detour and $k$-Longest Detour discussed above are significantly slower than the fastest known algorithms for the analogous $k$-Path and $k$-Longest Path problems. This motivates the questions: can $k$-Detour be solved as quickly as $k$-Path, and can $k$-Longest Detour be solved as quickly as $k$-Longest Path? Given the extensive and influential line of work that has gone into finding faster algorithms for $k$-Path and $k$-Longest Path, obtaining faster algorithms for these detour problems as well is an interesting open problem in parameterized complexity and exact algorithms.

## Our Results

The main result of our work is a faster algorithm for $k$-Detour on undirected graphs.

▶ **Theorem 1.** *In undirected graphs, $k$-Detour can be solved in $1.853^k \operatorname{poly}(n)$ time.*

This marks a significant improvement over the previous fastest $2.746^k \operatorname{poly}(n)$ time algorithm for $k$-Detour (and shows, for example, that this problem can be solved in faster than $2^k \operatorname{poly}(n)$ time, which is often a barrier for parameterized problems). Since the fastest known algorithms for $k$-Longest Detour over undirected graphs have a bottleneck of solving $2k$-Detour, Theorem 1 implies the following result.

▶ **Theorem 2.** *In undirected graphs, $k$-Longest Detour can be solved in $3.432^k \operatorname{poly}(n)$ time.*

Again, this is a significant improvement over the previous fastest algorithm for $k$-Longest Detour on undirected graphs, which ran in $7.539^k \operatorname{poly}(n)$ time.

Our algorithm for Theorem 1 applies the fact that $k$-Path is easier to solve on undirected graphs which have a prescribed vertex partition into two sets, where we constrain the path to contain a particular number of nodes from one set, and a particular number of edges

[1] Given an instance of $k$-Path, add an edge from $s$ to $t$. Then a solution to $(k-1)$-Detour in this new graph corresponds to a solution to $k$-Path in the original graph.

whose vertices are in the other set. Formally, we consider the $(\ell, k_1, \ell_2)$-Bipartitioned Path problem: given a graph $G$ on $n$ nodes, whose vertices are partitioned into parts $V_1$ and $V_2$, with distinguished vertices $s$ and $t$, the goal is to determine if $G$ contains a simple path from $s$ to $t$ of length $\ell$, which uses exactly $k_1$ vertices from $V_1$, and exactly $\ell_2$ edges whose endpoints are both in $V_2$. A careful application of the following result from [3] is the main source of the speed-up in our algorithm for $k$-Detour.

▶ **Lemma 3** ([3, Section 2]). *Let $\ell, k_1, \ell_2$ be nonnegative integers satisfying the inequality $\ell + 1 \geq k_1 + 2\ell_2$. Then over undirected graphs, the $(\ell, k_1, \ell_2)$-Bipartitioned Path problem can be solved in $2^{k_1+\ell_2}\,\mathrm{poly}(n)$ time.*

Although this "Bipartitioned Path" problem may appear esoteric at first, Lemma 3 plays a crucial role in obtaining the fastest known algorithm for $k$-Path in undirected graphs [3], and an analogue of Lemma 3 for paths of length at least $k$ is the basis for the fastest known algorithm for $k$-Longest Path in undirected graphs [7]. Proofs of Lemma 3 can be found in [3, Section 2], [5, Section 10.4], and in the full version of this paper in [1, Appendix B].

In Section 3, we provide an intuitive overview of how Lemma 3 helps us obtain our algorithm for $k$-Detour.

The fastest known algorithms for the path and detour problems discussed above all use randomness. Researchers are also interested in obtaining fast *deterministic* algorithms for these problems. We note that a simplified version of our algorithm for $k$-Detour implies faster deterministic algorithms for these detour problems over undirected graphs.

▶ **Theorem 4.** *The $k$-Detour problem can be solved over undirected graphs by a deterministic algorithm in $4.082^k\,\mathrm{poly}(n)$ time.*

Prior to this work, the fastest known deterministic algorithm for $k$-Detour on undirected graphs ran in $6.523^k\,\mathrm{poly}(n)$ time [2].

▶ **Theorem 5.** *The $k$-Longest Detour problem can be solved over undirected graphs by a deterministic algorithm in $16.661^k\,\mathrm{poly}(n)$ time.*

Prior to this work, the fastest known deterministic algorithm for $k$-Longest Detour on undirected graphs ran in $42.549^k\,\mathrm{poly}(n)$ time [8].

In summary, we obtain new randomized and deterministic algorithms for $k$-Detour and $k$-Longest Detour over undirected graphs, whose runtimes present significant advances over what was previously known for these problems.

## Organization

The remainder of this paper presents our new algorithms $k$-Detour. A thorough discussion of additional related work can is included in the full version of this paper in [1, Appendix A].

In Section 2, we introduce relevant notation, assumptions, and definitions concerning graphs. In Section 3, we provide an overview of our algorithm for $k$-Detour. In Section 4, we present the details of our algorithm, and prove its correctness. The runtime analysis for our algorithm (and thus the formal proofs of Theorems 1, 2, 4, and 5, given correctness of our algorithm) are presented in Section 5. In Section 6, we highlight some open problems.

## 2 Preliminaries

Given positive integers $a$ and $b$, we let $[a] = \{1, 2, \ldots, a\}$, and $[a, b] = \{a, a+1, \ldots, b\}$. Given an integer $a$ and a set of integers $S$, we define $a + S = \{a + s \mid s \in S\}$.

Throughout, we let $G$ denote the input graph. We assume that $G$ is undirected, has vertex set $V$ with $|V| = n$, and, without loss of generality, is connected.[2] Throughout, we let $s$ and $t$ denote the two distinguished vertices that come as part of the input to the $k$-Detour problem. Given a vertex $u$, we let $d(u) = \mathsf{dist}(s, u)$ denote its distance from $s$. This distance is well-defined, since $G$ is connected. Given a path $P$ containing vertices $u$ and $v$, we let $P[u, v]$ denote the subpath from $u$ to $v$ on $P$.

Given an edge $e = (u, v)$ from $u$ to $v$, we say $e$ is ***forward*** if $d(v) = d(u) + 1$, ***backwards*** if $d(v) = d(u) - 1$, and ***stable*** if $d(v) = d(u)$. In an undirected graph, by triangle inequality and symmetry of distance, adjacent vertices $u$ and $v$ have $|d(u) - d(v)| \leq 1$, so every edge in a path falls into one of these three categories.

Given two vertices $u, v \in V$, let $G_{(u,v]}$ denote the induced subgraph of $G$ on the set $\{u\} \cup \{w \in V \mid d(u) < d(w) \leq d(v)\}$. Let $G_{(u,\infty)}$ denote the induced subgraph of $G$ on the set $\{u\} \cup \{w \in V \mid d(u) < d(w)\}$. Note that for every $u$ and $v$, the subgraphs $G_{(u,v]}$ and $G_{(v,\infty)}$ overlap at vertex $v$, but are disjoint otherwise.

# 3 Technical Overview

In this section, we provide an overview of how our $k$-Detour algorithm works. Our starting point is the algorithm for this problem presented in [2, Section 4], which we review in Section 3.1. Then in Section 3.2 we review how the algorithm from Lemma 3 for $(\ell, k_1, \ell_2)$-Bipartitioned Path has previously been used to obtain the fastest known algorithm for $k$-Path in undirected graphs. With this context established, in Section 3.3 we outline how we combine the techniques from Sections 3.1 and 3.2 with new ideas to prove Theorem 1.

## 3.1 Previous Detour Algorithm

The previous algorithm for $k$-Detour from [2, Section 4] performs dynamic programming over nodes in the graph, starting from $t$ and moving to vertices closer to $s$. In the dynamic program, for each vertex $x$ with $d(x) \leq d(t)$, we compute all offsets $r \leq k$ such that there is a path of length $d(t) - d(x) + r$ from $x$ to $t$ in the subgraph $G_{(x,\infty)}$. Determining this set of offsets for $x = s$ solves the $k$-Detour problem, since $G_{(s,\infty)} = G$.

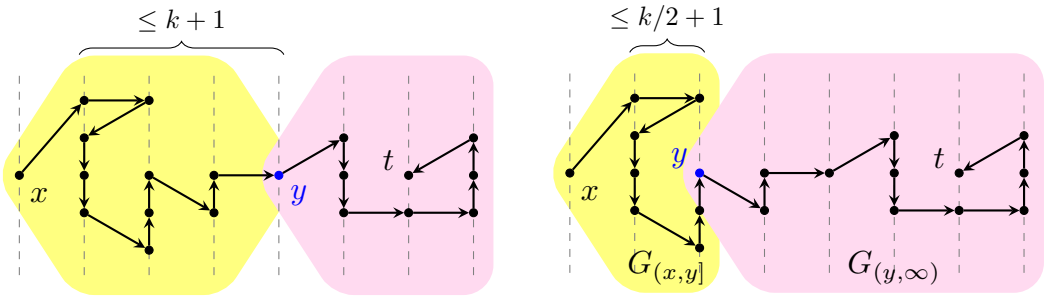
If $d(t) - d(x) \leq k$, we can find all such offsets just by solving $\ell$-Path for $\ell \leq 2k$.

So, suppose we are given a vertex $x$ with $d(t) - d(x) \geq k + 1$ and an offset $r \leq k$, and wish to determine if there is a path of length $d(t) - d(x) + r$ from $x$ to $t$ in $G_{(x,\infty)}$. If there is such a path $P$, then [2] argues that $P$ can always be split in as depicted in Figure 1a: for some vertex $y$ with $d(y) > d(x)$, we can decompose $P$ into two subpaths:

1. a subpath $A$ from $x$ to $y$ of length at most $2k + 1$, such that all internal vertices $v$ in $A$ satisfy $d(x) < d(v) < d(y)$, and
2. a subpath $B$ from $y$ to $t$ in $G_{(y,\infty)}$ of length at most $d(t) - d(y) + k$.

We can always split a path $P$ in this manner because $P$ has length at most $d(t) - d(x) + k$, so at most $k$ edges in $P$ are not forward edges. Intuitively, as we follow the vertices along the path $P$, the distance of the current vertex from $s$ can decrease or stay the same at most $k$ times, and so $P$ cannot contain too many vertices which are the same distance from $s$. This allows one to argue that there is a vertex $y$ with $d(y) \leq d(x) + k + 1$ such that all internal vertices $v$ of the subpath $P[x, y]$ have $d(x) < d(v) < d(y)$. Since $d(y) \leq d(x) + k + 1$ and $P$ has length at most $d(t) - d(x) + k$, it turns out that $P[x, y]$ has length at most $2k + 1$.

[2] If $G$ were not connected, we could replace it with the connected component containing $s$, and solve the detour problems on this smaller graph instead.




**(a) Previous Algorithms**: A subpath $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of a solution path can always be split at a vertex $y$ with $d(y) \leq d(x) + k + 1$, such that $d(u) \neq d(y)$ for all vertices $u \neq y$ in $P$.

**(b) Our Algorithm**: In undirected graphs, a subpath $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of a solution path can be split at a vertex $y$ with $d(y) \leq d(x)+k/2+1$, such that $P[x, y]$ is in $G_{(x,y]}$ and $P[y, t]$ is in $G_{(y,\infty)}$.

**Figure 1** To solve $k$-Detour, we split subpaths $P$ of candidate solutions at a vertex $y$ satisfying certain nice properties. We obtain a speed-up by getting better upper bounds on $d(y)$ in Figure 1b than previous work did in Figure 1a, by allowing $P[x, y]$ to have internal vertices $u$ with $d(u) = d(y)$.

Note that since $G_{(y,\infty)}$ only contains vertices $v$ with $d(v) \geq y$, the paths $A$ and $B$ must be disjoint. We can find the length of $A$ using an algorithm for $(2k+1)$-Path, and the length of $B$ will have already been computed in our dynamic program (since $y$ is further from $s$ than $x$). So, by trying out all possible $y$, finding the possible lengths for subpaths $A$ and $B$, and then adding up these lengths, we can get all possible lengths for $P$ in the dynamic program, and solve $k$-Detour.

## 3.2 Previous Path Algorithm

The fastest known algorithm for $k$-Path in undirected graphs goes through the $(k, k_1, \ell_2)$-Bipartitioned Path problem. Recall that in this problem, we are given a bipartition $V_1 \sqcup V_2$ of the vertices in the graph, and want to find a path of length $k$ from $s$ to $t$, which uses $k_1$ vertices in $V_1$ and $\ell_2$ edges with both endpoints in $V_2$. The authors of [3] showed that $(k, k_1, \ell_2)$-Bipartitioned Path can be solved in $2^{k_1+\ell_2}\,\mathrm{poly}(n)$ time over undirected graphs.

Why does this imply a faster algorithm for $k$-Path in undirected graphs? Well, suppose the input graph contains a path $P$ of length $k$ from $s$ to $t$. Consider a uniform random bipartition of the vertices of the graph into parts $V_1$ and $V_2$. We expect $(k+1)/2$ vertices of $P$ to be in $V_1$, and $k/4$ edges of $P$ to have both endpoints in $V_2$. In fact, this holds with constant probability, so we can solve $k$-Path by solving $(k, (k+1)/2, k/4)$-Bipartitioned Path in the randomly partitioned graph. By Lemma 3 this yields a $2^{3k/4}\,\mathrm{poly}(n) \approx 1.682^k\,\mathrm{poly}(n)$ time algorithm for $k$-Path. We can obtain a faster algorithm using the following modification: take several uniform random bipartitions of the graph, and solve $(k, k_1, \ell_2)$-Bipartitioned Path separately for each bipartition, for $k_1 + \ell_2 \leq 3(1-\varepsilon)k/4$, where $\varepsilon > 0$ is some constant. The number of bipartitions used is some function of $k$ and $\varepsilon$, set so that with high probability at least one of the partitions $V_1 \sqcup V_2$ has the property that the total number of vertices of $P$ in $V_1$ and number of edges of $P$ with both endpoints in $V_2$ is at most $3(1-\varepsilon)k/4$. Setting the parameter $\varepsilon$ optimally yields a $1.657^k\,\mathrm{poly}(n)$ time algorithm for $k$-Path [3].

## 3.3 Our Improvement

As in the previous approach outlined in Section 3.1, our algorithm for $k$-Detour performs dynamic programming over vertices in the graph, starting at $t$, and then moving to vertices closer to $s$. For each vertex $x$ with $d(x) \leq d(t)$, we compute all offsets $r \leq k$ such that there is a path of length $d(t) - d(x) + r$ from $x$ to $t$ in the subgraph $G_{(x,\infty)}$. Obtaining this information for $x = s$ and $r = k$ solves the $k$-Detour problem.

Given a vertex $x$ and offset $r \leq k$, we wish to determine if $G$ contains a path of length $d(t) - d(x) + r$ from $x$ to $t$ in $G_{(x,\infty)}$. Suppose there is such a path $P$. If $d(t) - d(x)$ is small enough, it turns out we can find $P$ by solving $p$-Path for small values of $p$. So, for the purpose of this overview, suppose that $d(t) - d(x)$ is sufficiently large. In this case, as outlined in Section 3.1, previous work showed that $P$ can be split into two subpaths $A$ and $B$ contained in disjoint subgraphs, such that $A$ has length at most $2k + 1$. This splitting argument holds even for directed graphs. Our first improvement comes from the observation that in undirected graphs, we can decompose the path $P$ with a smaller prefix: as depicted in Figure 1b, there must exist a vertex $y$ with $d(y) > d(x)$, such that $P$ splits into a subpath $A$ from $x$ to $y$ in $G_{(x,y]}$ of length at most $3k/2+1$, and a path $B$ from $y$ to $t$ in $G_{(y,\infty)}$ of length at most $d(t) - d(y) + k$. We can find the length of $A$ by solving $(3k/2 + 1)$-Path, and the length of $B$ will already have been computed by dynamic programming, since $d(y) > d(x)$.

This split is possible because any consecutive vertices $u$ and $v$ in $P$ have $|d(u) - d(v)| \leq 1$ (this is true for undirected graphs, but is not true in general for directed graphs). Since $P$ has length at most $d(t) - d(x) + k$, it turns out that $P$ has at most $k/2$ backwards edges. This lets us argue that there exists a vertex $y$ with $d(y) \leq d(x) + k/2 + 1$ such that $P[x, y]$ is contained in $G_{(x,y]}$ and $P[y, t]$ is contained in $G_{(y,\infty)}$. Finally, $A = P[x, y]$ should have length at most $k$ more than $d(y) - d(x)$, which means it has length at most $3k/2 + 1$.

This simple modification already yields a faster algorithm[3] for $k$-Detour. We get further improvements by performing casework on the number of stable edges in $P$ (recall that an edge $(u, v)$ is stable if both its endpoints have the same distance $d(u) = d(v)$ from $s$).

First, suppose $P$ has at least $m$ stable edges, for some parameter $m$. Since $P$ has length at most $d(t) - d(x) + k$, we can argue that $P$ has at most $(k - m)/2$ backwards edges. With this better upper bound on the number of backwards edges, we can improve the splitting argument and show that $P$ decomposes into subpaths $A$ and $B$, such that the length of $A$ is at most $(3k - m)/2$, and the length of $B$ was already computed by our dynamic program. It then suffices to solve $(3k - m)/2$-Path, which yields a speed-up whenever $m \geq \Omega(k)$.

Otherwise, $P$ has at most $m$ stable edges. In this case, we consider the bipartition $V_1 \sqcup V_2$ of the vertex set, where $V_1$ has all vertices at an odd distance from $s$, and $V_2$ has all vertices with even distance from $s$. Since $G$ is undirected, consecutive vertices on the path $P$ differ in their distance from $s$ by at most one. In particular, all forward and backward edges in $P$ cross between the parts $V_1$ and $V_2$. Only the stable edges can contribute to edges with both endpoints in $V_2$. Since we assumed that the number of stable edges is small, it turns out we can find the length of the subpath $A$ of $P$ by solving $(\ell, k_1, \ell_2)$-Bipartitioned Path with respect to the given bipartition, for some $\ell_2$ which is very small. In particular, this approach computes the length of $A$ faster than naively solving $(3k - m)/2$-Path. By setting an appropriate threshold for $m$, we can minimize the runtimes of the algorithm in both of the above cases, and establish Theorem 1.

So in summary, our faster algorithms come from two main sources of improvement: using the structure of shortest paths in undirected graphs to get a better "path-splitting" argument in the dynamic program from $k$-Detour, and cleverly applying the fast algorithm from Lemma 3 for $(\ell, k_1, \ell_2)$-Bipartitioned Path with carefully chosen bipartitions.

We note that our application of $(\ell, k_1, \ell_2)$-Bipartitioned Path is qualitatively different from its uses in previous work. As discussed in Section 3.1, previous algorithms for $k$-Detour work by solving instances of $k$-Path, and as described in Section 3.2, the fastest algorithms for $k$-Path on undirected graphs work by reduction to various instances of $(\ell, k_1, \ell_2)$-Bipartitioned Path. Thus, previous algorithms for $k$-Detour on undirected graphs *implicitly* rely on the

[3] In fact, this observation already yields the fast deterministic algorithms for Theorems 4 and 5.

fast algorithm for $(\ell, k_1, \ell_2)$-Bipartitioned Path, applied to random bipartitions of the input graph. We obtain a faster algorithm for $k$-Detour arguing that in certain cases, we can "beat randomness," by constructing bipartitions which leverage structural information about the graph (namely, whether the shortest path distance from $s$ to a given vertex is even or odd).

## 4 Detour Algorithm

In this section, we present Algorithm 1, our new algorithm for the $k$-Detour problem. As mentioned in the previous section, our algorithm behaves differently depending on the number of stable edges that a potential solution path contains. In particular, the algorithm depends on a parameter $\alpha \in (0, 1)$, which determines the threshold for what counts as "many" stable edges. Later, we will set $\alpha$ to optimize the runtime of Algorithm 1. Certain lines of Algorithm 1 have comments indicating case numbers, which are explained in Section 4.1.

Our algorithm computes a set $L(x)$ for each vertex $x$ in the graph, corresponding to the possible lengths of potential subpaths from $x$ to $t$ of a solution path from $s$ to $t$.

In step 3 of Algorithm 1, we compute $L(x)$ for all $x$ that are "far" from $s$, by solving instances of $\ell$-Path for $\ell \leq (3-\alpha)k/2$. Starting in step 4, we compute $L(x)$ for vertices $x$ closer to $s$, in terms of the previously computed sets $L(y)$ for vertices $y$ further from $s$. In steps 5 through 7, we compute some lengths in $L(x)$ by solving instances of $(a, k_1, \ell_2)$-Bipartitioned Path for appropriate $a, k_1, \ell_2$ values, and in 8 and 9 we compute the remaining lengths in $L(x)$ by solving $a$-Path for $a \leq (3 - \alpha)k/2 + 1$.

### 4.1 Correctness

In this section, we show that Algorithm 1 correctly solves the $k$-Detour problem for any choice of $\alpha \in (0, 1)$. The main technical part of the proof lies in inductively showing that every possible solution path from $s$ to $t$ will be considered by the algorithm and its length will be included in the set $L(s)$. In Algorithm 1, we try out values of the variable $m$ from 0 to $k$, and execute differently depending on how $m$ compares to $\alpha k$. This is interpreted as follows: suppose there is a solution path $P$ from $x$ to $t$, then $m$ corresponds to a guess of the number of stable edges in $P$.

In **Case 1**, we guess that $P$ has few stable edges $m < \alpha k$ which corresponds to steps 5 to 7. Under **Case 1**, there are two possible structures a potential solution path might take on depending on how $d(x)$ compares to $d(t)$. We refer to the case where $d(x) - d(t)$ is small as **Case 1(a)** considered by step 6, and the case where $d(x) - d(t)$ is large as **Case 1(b)** considered by step 7. In **Case 2**, we guess that $m \geq \alpha k$, so $P$ has many stable edges, which corresponds to steps 8 to 9. These cases are also formally defined in our proof of correctness.

▶ **Theorem 6.** *For any fixed $\alpha \in (0, 1)$, Algorithm 1 correctly solves the $k$-Detour problem.*

**Proof.** We prove that upon halting, each set $L(x)$ computed by Algorithm 1 has the property that for all integers $\ell \in [d(t) - d(x), d(t) - d(x) + k]$, we have

$$\ell \in L(x) \text{ if and only if there is a path of length } \ell \text{ from } x \text{ to } t \text{ in } G_{(x,\infty)}. \tag{1}$$

If this property holds, then step 10 of Algorithm 1 returns the correct answer to the $k$-Detour problem, since $\mathsf{dist}(s, t) + k$ is in $L(s)$ if and only if there is a path from $s$ to $t$ of length $\mathsf{dist}(s, t) + k$ in $G_{(s,\infty)} = G$.

So, it suffices to show that Equation (1) holds for all vertices $x$. We prove this result by induction on the distance of $x$ from $s$ in the graph.

**Algorithm 1** Our algorithm for solving $k$-Detour in undirected graphs.

**Input:** An undirected graph $G$ with distinguished vertices $s$ and $t$, and a parameter $\alpha \in (0,1)$.

1: Initialize $V_1 \leftarrow \{x \in V \mid d(x) \equiv 1 \mod 2\}$, $V_2 \leftarrow \{x \in V \mid d(x) \equiv 0 \mod 2\}$.
2: For each vertex $x$ in the graph with $d(x) \leq d(t)$, initialize $L(x) \leftarrow \emptyset$.
▷ *$L(x)$ will be the set of lengths $\ell \in [d(t)-d(x), d(t)-d(x)+k]$ such that there is a path of length $\ell$ from $x$ to $t$ in $G_{(x,\infty)}$.*

3: For each vertex $x$ with $d(x) \in [d(t)-(1-\alpha)k/2, d(t)]$, set $L(x)$ to be the set of all positive integers $\ell \leq (3-\alpha)k/2$ such that there is a path of length $\ell$ from $x$ to $t$ in $G_{(x,\infty)}$.
▷ *Base case: we compute $L(x)$ for the vertices $x$ which are furthest from $s$.*

4: For each $d$ from $d(t)-(1-\alpha)k/2-1$ down to 0, for each vertex $x$ with $d(x)=d$, complete steps 5 through 9.
▷ *Inductive Case: compute $L(x)$ layer by layer towards $s$.*

5: For each integer $m$ with $0 \leq m < \alpha k$, and for each choice of integers $k_1, \ell_2 \geq 0$ satisfying $k_1 + \ell_2 \leq (3k+m+2)/4$, complete steps 6 and 7.
▷ *This step handles **Case 1:** the solution path has few stable edges.*

6: If there is a path of length $\ell \leq 2k_1 + \ell_2$ from $x$ to $t$ in $G_{(x,\infty)}$, update $L(x) \leftarrow L(x) \cup \{\ell\}$.
▷ *This step handles **Case 1(a):** $d(t)-d(x) \leq (k-m)/2$.*

7: Try out all vertices $y$ with $d(y) \in [d(x)+1, \min(d(t), d(x)+(3k-m)/2+1)]$. If for some such $y$, there is a path from $x$ to $y$ in $G_{(x,y]}$ of length $a \leq 2k_1+\ell_2$ with exactly $k_1$ vertices in $V_1$, and $\ell_2$ edges with both endpoints in $V_2$, update $L(x) \leftarrow L(x) \cup (a + L(y))$.
▷ *This step handles **Case 1(b):** $d(t)-d(x) > (k-m)/2$.*

8: For each integer $m$ with $\alpha k < m \leq k$, complete step 9.
▷ *This step handles **Case 2:** the solution path has many stable edges.*

9: Try out all vertices $y$ with $d(y) \in [d(x)+1, d(x)+(1-\alpha)k/2+1]$. If for some such $y$, there is a path from $x$ to $y$ in $G_{(x,y]}$ of length $a \leq (3-\alpha)k/2+1$, update $L(x) \leftarrow L(x) \cup (a + L(y))$

10: Return **yes** if and only if $(\mathsf{dist}(s,t)+k) \in L(s)$.

**Base Case.** For the base case, suppose $x$ is a vertex with

$$d(x) \in [d(t)-(1-\alpha)k/2, d(t)]. \tag{2}$$

Then $L(x)$ is computed in step 3 of Algorithm 1. We now verify that Equation (1) holds.
First, suppose $\ell \in L(x)$.
Then, $\ell$ must be the length of some path from $x$ to $t$ in $G_{(x,\infty)}$ by design.
Conversely, suppose we have a path $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of some length

$$\ell \leq d(t)-d(x)+k.$$

Then by the assumption on $x$ from Equation (2) in this case, we have

$$\ell \leq d(t) - d(x) + k \leq (1-\alpha)k/2 + k = (3-\alpha)k/2$$

so step 3 of Algorithm 1 correctly includes $\ell$ in $L(x)$.

Thus Equation (1) holds for all vertices $x$ satisfying Equation (2).

**Inductive Case.** For the inductive step, suppose $x$ is a vertex with

$$d(x) \leq d(t) - (1-\alpha)k/2 - 1. \tag{3}$$

We may inductively assume that we have computed sets $L(y)$ satisfying Equation (1), for all vertices $y$ with $d(y) > d(x)$.

Suppose $\ell \in L(x)$ at the end of Algorithm 1. Then either $\ell$ was added to $L(x)$ in step 6, or $\ell$ was added to $L(x)$ in steps 7 or 9 of Algorithm 1. In the former case, $\ell$ is the length of a path from $x$ to $t$ in $G_{(x,\infty)}$ by design. In the latter cases, we have $\ell = a + b$, where $a$ is the length of some path from $x$ to $y$ (for some vertex $y$ with $d(y) > d(x)$) in $G_{(x,y]}$, and (by the inductive hypothesis) $b$ is the length of some path from $y$ to $t$ in $G_{(y,\infty)}$. Since $G_{(x,y]}$ and $G_{(y,\infty)}$ intersect only at $y$, the union of these paths is a path from $x$ to $t$ in $G_{(t,\infty)}$. So, every integer in $L(x)$ is a valid length of a path from $x$ to $t$ in $G_{(x,\infty)}$ as desired.

Conversely, suppose there is a path $P$ from $x$ to $t$ in $G_{(x,\infty)}$ of length

$$\ell \in [d(t) - d(x), d(t) - d(x) + k]. \tag{4}$$

We prove that $\ell$ appears in $L(x)$.

To do this, we will analyze the number of forward, backward, and stable edges appearing in $P$. Note that $P$ has at least $d(t) - d(x)$ forward edges, since $P$ begins at a vertex at distance $d(x)$ from $s$, ends at a vertex at distance $d(t)$ from $s$, and only the forward edges allow us to move to vertices further from $s$.

Let $m$ denote the number of stable edges in $P$. We have $m \leq k$, since the length of $P$ is at most $d(t) - d(x) + k$, and $P$ has at least $d(t) - d(x)$ forward edges.

▷ Claim 7. Suppose $d(x) \leq d(t) - (k-m)/2 - 1$. Then $P$ contains a vertex $y$ such that

1. $d(y) \in [d(x) + 1, d(x) + (k-m)/2 + 1]$,
2. every vertex $u \in P[y,t]$ with $u \neq y$ has $d(u) > d(y)$, and
3. every vertex $v \in P[x,y]$ has $d(v) \leq d(y)$.

Proof. For each $i \in [(k-m)/2 + 1]$, let $z_i$ denote the last vertex on $P$ satisfying

$$d(z_i) = d(x) + i.$$

These vertices exist because we are assuming that $d(x) \leq d(t) - (k-m)/2 - 1$, and $P$ must contain vertices $v$ with $d(v) = d$ for every $d \in [d(x), d(t)]$.

By definition, each $z_i$ satisfies conditions **1** and **2** from the claim. If some $z_i$ satisfies condition **3** as well, then the claim is true.

So, suppose that none of the $z_i$ satisfy condition **3**. This means that for each index $i$, the subpath $P[x, z_i]$ contains a vertex $u$ with $d(u) > d(z_i)$. Consecutive vertices in $P$ differ in their distance from $s$ by at most one, so $P[x, z_i]$ must contain an edge $e = (v, w)$ such that $d(v) = d(w) + 1$ and $d(w) = d(z_i) = d(x) + i$. That is, $P$ contains a backwards edge from a vertex at distance $i + 1$ from $s$ to a vertex at distance $i$ from $s$, as depicted in Figure 2.

Note that $z_1, z_2, \ldots, z_{(k-m)/2+1}$ occur on $P$ in the listed order. This is because

$$d(z_1) < d(z_2) < \cdots < d(z_{(k-m)/2+1})$$

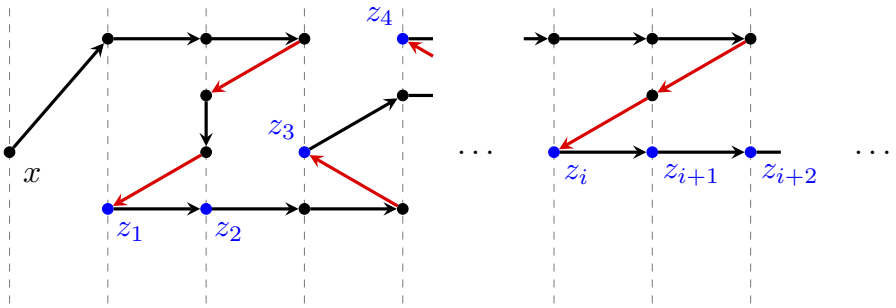



**Figure 2** If node $z_i$ does not satisfy condition **3** of Claim 7, it means that before hitting $z_i$, the path visited a node further from $s$ than $z_i$. Thus, we can associate $z_i$ with some backwards edge. The presence of too many of these backwards edges would violate the length condition on $P$, so it turns out that one such node (in the figure, $z_{i+2}$) does have to satisfy condition **3**.

and each $z_i$ satisfies condition **1** from the claim. Combined with the discussion from the previous paragraph, this means that $P$ contains at least $(k-m)/2+1$ backwards edges. We now argue that this violates the assumption on the length of $P$.

Let $f$ and $b$ denote the number of forward and backwards edges in $P$ respectively.

Since $P$ starts at $x$ and ends at $t$, we have $f - b = d(t) - d(x)$, which implies that

$$f = d(t) - d(x) + b. \tag{5}$$

Then the total length of $P$ is $f + b + m = d(t) - d(x) + m + 2b$ by Equation (5). However, since $P$ has at least $(k-m)/2+1$ backwards edges, this length satisfies

$$d(t) - d(x) + m + 2b \geq d(t) - d(x) + m + 2\left((k-m)/2 + 1\right) > d(t) - d(x) + k$$

which contradicts the fact that the length $\ell$ of $P$ satisfies Equation (4). Thus our assumption was incorrect, and one of the $z_i$ satisfies all three conditions from the claim, as desired. ◁

We now perform casework on the number of stable edges $m$ in $P$. We start with **Case 2** from step 8 of Algorithm 1, since this is the easiest case to analyze.

**Case 2: Many Stable Edges ($m \geq \alpha k$).** Suppose $m \geq \alpha k$. In this case, by Equation (3) we have

$$d(x) \leq d(t) - (1-\alpha)k/2 - 1 \leq d(t) - (k-m)/2 - 1$$

So, by Claim 7, there exists a vertex $y$ in $P$ satisfying the three conditions of Claim 7.

By condition **3** from Claim 7, the subpath $A = P[x, y]$ is contained in $G_{(x,y]}$. By condition **2** from Claim 7, the subpath $B = P[y, t]$ is contained in $G_{(y,\infty)}$.

Let $a$ denote the length of $A$, and $b$ denote the length of $B$.

Since $A$ has length at least $d(y) - d(x)$, and $P$ has length at most $d(t) - d(x) + k$ by Equation (4), we know that the length $B$ satisfies

$$b \leq d(t) - d(y) + k. \tag{6}$$

By the inductive hypothesis, $L(y)$ satisfies Equation (1), so $b \in L(y)$.

Similar to the reasoning that established Equation (6), we can prove that

$$a \leq d(y) - d(x) + k. \tag{7}$$

By condition **1** of Claim 7, we know that $d(y) \leq d(x) + (k-m)/2 + 1$. Since $m \geq \alpha k$, this implies that $d(y) \leq d(x) + (1-\alpha)k/2 + 1$.

Substituting this into Equation (7) yields

$$a \leq (1-\alpha)k/2 + 1 + k = (3-\alpha)k/2 + 1.$$

Thus, the length $a$ of $A$ will be found in step 9 of Algorithm 1. As mentioned before, $b \in L(y)$. Thus, $\ell = a + b \in (a + L(y))$ is correctly added to the set $L(x)$ in step 9 of Algorithm 1, which proves the desired result in this case.

**Case 1: Few Stable Edges ($m < \alpha k$).** If we do not fall into **Case 2**, we must have $m < \alpha k$. Recall that in step 1 of Algorithm 1, we defined $V_1 = \{u \mid d(u) \text{ is odd}\}$ and $V_2 = \{u \mid d(u) \text{ is even}\}$.

We want to argue that most edges in path $P$ cross the bipartition $V_1 \sqcup V_2$. To that end, the following claim will be helpful.

▷ Claim 8. Let $Q$ be a path of length $q$, with at most $m$ stable edges. Let $k_1$ denote the number of vertices of $Q$ in $V_1$, and let $\ell_2$ denote the number of edges in $Q$ with both endpoints in $V_2$. Then we have

$$k_1 + \ell_2 \leq (q + m + 1)/2.$$

Proof. Let $k_2$ denote the number of vertices of $Q$ in $V_2$.

Consider the cycle $C$ formed by taking $Q$ together with an additional edge between its endpoints (this new edge is imagined for the purpose of argument, and does not change the definition of $V_1$ and $V_2$).

Let $q_1$, $q_2$, and $q_{\text{cross}}$ denote the number of edges of $C$ with both endpoints in $V_1$, both endpoints in $V_2$, and endpoints in both $V_1$ and $V_2$ respectively. We have

$$2k_1 = 2q_1 + q_{\text{cross}} \tag{8}$$

because both sides of the above equation count the number of pairs $(u, e)$ such that $u$ is a vertex in $C \cap V_1$, and $e$ is an edge in $C$ incident to $u$. A symmetric argument implies that

$$2q_2 + q_{\text{cross}} = 2k_2. \tag{9}$$

Adding Equation (8) and Equation (9) together and simplifying yields

$$k_1 + q_2 = k_2 + q_1.$$

This implies that

$$k_1 + q_2 = (k_1 + k_2 + q_1 + q_2)/2.$$

Since $C$ is $Q$ with one additional edge, we have $\ell_2 \leq q_2$. So the above equation implies that

$$k_1 + \ell_2 \leq (k_1 + k_2 + q_1 + q_2)/2. \tag{10}$$

We have

$$k_1 + k_2 = q + 1 \tag{11}$$

since the total number of vertices in $Q$ must be one more than its length. By assumption on the number of stable edges in $Q$, we have

$$q_1 + q_2 \leq m. \tag{12}$$

Substituting Equation (11) and Equation (12) into the right hand side of Equation (10) yields

$$k_1 + \ell_2 \leq (q + m + 1)/2$$

which proves the desired result. ◁

With Claim 8 established, we are now ready to analyze the two subcases under **Case 1**, based on the relative distances of $x$ and $t$ from $s$.

**Case 1(a): $d(t) - d(x)$ is small.** Suppose that $d(x) \in [d(t) - (k - m)/2, d(t)]$.

In this case, Equation (4) implies that $P$ has length

$$\ell \leq d(t) - d(x) + k \leq (3k - m)/2.$$

Let $k_1$ denote the number of vertices of $P$ in $V_1$, and $k_2$ denote the number of edges in $P$ with both endpoints in $V_2$. Then by setting $Q = P$ and $q = \ell$ in Claim 8, we have

$$k_1 + \ell_2 \leq (\ell + m + 1)/2 \leq (3k + m + 2)/4. \tag{13}$$

Also, note that $P$ has length $\ell \leq 2k_1 + \ell_2$, since $2k_1$ is greater than or equal to the number of edges in $P$ incident to a vertex in $V_1$. This observation, together with Equation (13), shows that in this case, the length $\ell$ is correctly included in $L(x)$ in step 6 of Algorithm 1.

**Case 1(b): $d(t) - d(x)$ is large.** If we do not fall into **Case 1(a)**, it means that

$$d(x) \leq d(t) - (k - m)/2 - 1. \tag{14}$$

Thus, by Claim 7, there exists a vertex $y$ in $P$ satisfying the three conditions of Claim 7. The proof that $\ell \in L(x)$ in this case is essentially a combination of the proofs from **Case 2** and **Case 1(a)**.

As in **Case 2**, by condition **3** from Claim 7, the subpath $A = P[x, y]$ is contained in $G_{(x,y]}$. By condition **2** from Claim 7, the subpath $B = P[y, t]$ is contained in $G_{(y,\infty)}$.

Let $a$ and $b$ denote the lengths of paths $A$ and $B$ respectively. Reasoning identical to the arguments which established Equations (6) and (7) prove that in this case we also have

$$b \leq d(t) - d(y) + k \tag{15}$$

and

$$a \leq d(y) - d(x) + k. \tag{16}$$

Condition **1** of Claim 7 implies that $d(y) \leq d(x) + (k - m)/2 + 1$. Substituting this into Equation (16) implies that

$$a \leq (3k - m)/2 + 1.$$

Let $k_1$ denote the number of vertices of $A$ in $V_1$, and let $\ell_2$ denote the number of edges in $A$ with both endpoints in $V_2$. Then by setting $Q = A$ and $q = a$ in Claim 8, we have

$$k_1 + \ell_2 \leq (a + m + 1)/2 \leq (3k + m + 2)/4. \tag{17}$$

Also, we know that $a \leq 2k_1 + \ell_2$, because $2k_1$ is greater than or equal to the number of edges in $A$ incident to a vertex in $V_1$. Combining this observation with Equation (17), we see that the length $a$ is indeed computed in step 7 of Algorithm 1.

By the inductive hypothesis (Equation (1)) and Equation (15), we know that $b \in L(y)$. Thus we have $\ell = a + b \in (a + L(y))$, so in this case, $\ell$ is correctly included in $L(x)$ in step 7 of Algorithm 1.

This completes the induction, and proves that Equation (1) holds for all vertices $x$ in the graph. In particular, Equation (1) holds for $x$ equal to $s$. This implies that step 10 of Algorithm 1 returns the correct answer to the $k$-Detour algorithm. ◀

## 5 Applications

In this section, we present consequences of our new algorithm for $k$-Detour from Section 4.

▶ **Theorem 1.** *In undirected graphs, $k$-Detour can be solved in $1.853^k \operatorname{poly}(n)$ time.*

**Proof.** By Theorem 6, Algorithm 1 correctly solves $k$-Detour, for any value $\alpha \in (0, 1)$,

What is the runtime of Algorithm 1? Well, steps 3 and 9 of Algorithm 1 involve solving polynomially many instances of $\ell$-Path, for $\ell \leq (3 - \alpha)k/2 + 1$. Using the fastest known algorithm for $k$-Path in undirected graphs [3], these steps take

$$1.657^{(3-\alpha)k/2} \operatorname{poly}(n)$$

time. The remaining computationally intensive steps of Algorithm 1 occur in steps 6 and 7, which can be implemented by solving $\operatorname{poly}(n)$ instances of $(\ell, k_1, \ell_2)$-Bipartitioned Path, for $k_1 + \ell_2 < (3k + \alpha k + 2)/4$. By Lemma 3, these steps then take

$$2^{(3+\alpha)k/4} \operatorname{poly}(n)$$

time overall. Then by setting $\alpha = 0.55814$ to balance the above runtimes, we see that we can solve $k$-Detour over undirected graphs in

$$\left(1.657^{(3-\alpha)k/2} + 2^{(3+\alpha)k/4}\right) \operatorname{poly}(n) \leq 1.8526^k \operatorname{poly}(n)$$

time, as desired. ◀

▶ **Theorem 2.** *In undirected graphs, $k$-Longest Detour can be solved in $3.432^k \operatorname{poly}(n)$ time.*

**Proof.** The proof of [8, Corollary 2] shows that $k$-Longest Detour in undirected graphs reduces, in polynomial time, to solving $p$-Detour for all $p \leq 2k$ and $\operatorname{poly}(n)$ instances of $(3k/2)$-Longest Path on graphs with at most $n$ nodes.

The proof of Theorem 1 implies that $k$-Detour can be solved over undirected graphs in $1.8526^k \operatorname{poly}(n)$ time. Previous work in [7, Section 6.3] shows that $k$-Longest Path can be solved over undirected graphs in $1.657^k \operatorname{poly}(n)$ time. Combining these results together with the above discussion shows that $k$-Longest Detour can be solved over undirected graphs in

$$\left(1.8526^{2k} + 1.657^{3k/2}\right) \operatorname{poly}(n) \leq 3.432^k \operatorname{poly}(n)$$

time, as desired. ◀

▶ **Theorem 4.** *The $k$-Detour problem can be solved over undirected graphs by a deterministic algorithm in $4.082^k \operatorname{poly}(n)$ time.*

**Proof.** By Theorem 6, we can solve $k$-Detour over an undirected graph by running Algorithm 1 with parameter $\alpha = 0$. When $\alpha = 0$ in Algorithm 1, steps 5, 6, 7 never occur. In this case, the algorithm only needs to solve $\text{poly}(n)$ instances of $\ell$-Path, for $\ell \leq 3k/2 + 1$, in steps 3 and 9. Since $k$-Path can be solved deterministically in $2.554^k \text{poly}(n)$ time [15], this means that we can solve $k$-Detour deterministically in

$$2.554^{3k/2} \text{poly}(n) \leq 4.0817^k \text{poly}(n)$$

time, as desired. ◀

▶ **Theorem 5.** *The $k$-Longest Detour problem can be solved over undirected graphs by a deterministic algorithm in $16.661^k \text{poly}(n)$ time.*

**Proof.** The proof of [8, Corollary 2] shows that $k$-Longest Detour in undirected graphs reduces, in deterministic polynomial time, to solving $p$-Detour for $p \leq 2k$, and $\text{poly}(n)$ instances of $(3k/2)$-Longest Path on graphs with at most $n$ nodes.

The proof of Theorem 4 implies that $k$-Detour can be solved over undirected graphs deterministically in $4.0817^k \text{poly}(n)$ time. Previous work [9] shows that $k$-Longest Path can be solved deterministically in $4.884^k \text{poly}(n)$ time. Combining these results together with the above discussion shows that $k$-Longest Detour can be solved over undirected graphs deterministically in

$$\left(4.0817^{2k} + 4.884^{3k/2}\right) \text{poly}(n) \leq 16.661^k \text{poly}(n)$$

time, as desired. ◀

## 6 Conclusion

In this paper, we obtained faster algorithms for $k$-Detour and $k$-Longest Detour over undirected graphs. However, many mysteries remain surrounding the true time complexity of these problems. We highlight some open problems of interest, relevant to our work.

1. The most pertinent question: what is the true parameterized time complexity of $k$-Detour and $k$-Longest Detour? In particular, could it be the case that $k$-Detour can be solved as quickly as $k$-Path, and $k$-Longest Detour can be solved as quickly as $k$-Longest Path? No known conditional lower bounds rule out these possibilities.
2. The current fastest algorithm for $k$-Longest Path in directed graphs has a bottleneck of solving $2k$-Path. The current fastest algorithm for $k$-Detour in directed graphs has a bottleneck of solving $2k$-Path. Similarly, the fastest known algorithm[4] for $k$-Longest Detour in undirected graphs requires first solving $2k$-Detour. Is this parameter blow-up necessary? Could it be possible to solve these harder problems with parameter $k$ faster than solving these easier problems with parameter $2k$?
3. The speed-up in our results crucially uses a fast algorithm for the $(\ell, k_1, \ell_2)$-Bipartitioned Path problem in undirected graphs. In directed graphs no $(2-\varepsilon)^\ell \text{poly}(n)$ time algorithm appears to be known for this problem, for any constant $\varepsilon > 0$ and interesting range of parameters $k_1$ and $\ell_2$. Such improvements could yield faster algorithms for $k$-Detour in directed graphs. Can we get such an improvement? Also of interest: can we get a faster deterministic algorithm for $(\ell, k_1, \ell_2)$-Bipartitioned Path?

[4] In fact, even the recent alternate algorithm of [10] for $k$-Longest Detour requires solving $2k$-Detour first.

4. An easier version of the previous question, also raised in [7, Section 9.1]: can we solve $k$-Path in directed bipartite graphs in $(2-\varepsilon)^k \operatorname{poly}(n)$ time, for some constant $\varepsilon > 0$? In the unparameterized setting, the Hamiltonian Path ($k$-Path for $k = n$) problem admits several distinct algorithms running in $(2-\varepsilon)^n \operatorname{poly}(n)$ time in directed bipartite graphs. Specifically, [6] shows Hamiltonian Path in directed bipartite graphs can be solved in $1.888^n \operatorname{poly}(n)$ time, and [4] uses very different methods to solve this problem even faster in $3^{n/2} \operatorname{poly}(n)$ time.[5] We conjecture that the same speed-up is possible for $k$-Path, so that this problem can be solved over directed bipartite graphs in $3^{k/2} \operatorname{poly}(n)$ time.

## References


1 Shyan Akmal, Virginia Vassilevska Williams, Ryan Williams, and Zixuan Xu. Faster detours in undirected graphs, 2023. arXiv:2307.01781.

2 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding Detours is Fixed-Parameter Tractable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.54.

3 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *Journal of Computer and System Sciences*, 87:119–139, August 2017. doi:10.1016/j.jcss.2017.03.003.

4 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed Hamiltonicity and Out-Branchings via Generalized Laplacians. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 91:1–91:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.91.

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, 2015. doi:10.1007/978-3-319-21275-3.

6 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *Journal of the ACM*, 65(3):1–46, March 2018. doi:10.1145/3148227.

7 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving, 2023. doi:10.48550/arXiv.2304.02091.

8 Fedor V. Fomin, Petr A. Golovach, William Lochet, Danil Sagunov, Kirill Simonov, and Saket Saurabh. Detours in Directed Graphs. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.29.

9 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Long directed (s, t)-path: FPT algorithm. *Information Processing Letters*, 140:8–12, December 2018. doi:10.1016/j.ipl.2018.04.018.

10 Ashwin Jacob, Michał Włodarczyk, and Meirav Zehavi. Long directed detours: Reduction to 2-disjoint paths, 2023. doi:10.48550/arXiv.2301.06105.


[5] It is also known that sufficient improvements to algorithms for multiplying two $n \times n$ matrices together would imply that even the weighted version of Hamiltonian Path in directed bipartite graphs can be solved in $(2-\varepsilon)^n \operatorname{poly}(n)$ time, for some constant $\varepsilon > 0$ [14].

**11** Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008.

**12** Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. *ACM Transactions on Algorithms*, 12(3):1–18, May 2016. doi:10.1145/2885499.

**13** Burkhard Monien. How to find long paths efficiently. In *North-Holland Mathematics Studies*, volume 109, pages 239–254. Elsevier, 1985.

**14** Jesper Nederlof. Bipartite TSP in $o(1.9999^n)$ time, assuming quadratic time matrix multiplication. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. ACM, June 2020. doi:10.1145/3357713.3384264.

**15** Dekel Tsur. Faster deterministic parameterized algorithm for k-path. *Theoretical Computer Science*, 790:96–104, October 2019. doi:10.1016/j.tcs.2019.04.024.

**16** Ryan Williams. Finding paths of length $k$ in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, February 2009. doi:10.1016/j.ipl.2008.11.004.

**17** Meirav Zehavi. A randomized algorithm for long directed cycle. *Information Processing Letters*, 116(6):419–422, June 2016. doi:10.1016/j.ipl.2016.02.005.

# A Local-To-Global Theorem for Congested Shortest Paths

**Shyan Akmal**
MIT, EECS and CSAIL, Cambridge, MA, USA

**Nicole Wein**
DIMACS, Rutgers University, Piscataway, NJ, USA

## Abstract

Amiri and Wargalla proved the following local-to-global theorem about shortest paths in directed acyclic graphs (DAGs): if $G$ is a weighted DAG with the property that for each subset $S$ of 3 nodes there is a shortest path containing every node in $S$, then there exists a pair $(s, t)$ of nodes such that there is a shortest $st$-path containing every node in $G$. We extend this theorem to general graphs. For undirected graphs, we prove that the same theorem holds (up to a difference in the constant 3). For directed graphs, we provide a counterexample to the theorem (for any constant). However, we prove a *roundtrip* analogue of the theorem which guarantees there exists a pair $(s, t)$ of nodes such that every node in $G$ is contained in the union of a shortest $st$-path and a shortest $ts$-path.

The original local-to-global theorem for DAGs has an application to the $k$-Shortest Paths with Congestion $c$ ($(k, c)$-SPC) problem. In this problem, we are given a weighted graph $G$, together with $k$ node pairs $(s_1, t_1), \ldots, (s_k, t_k)$, and a positive integer $c \leq k$, and tasked with finding a collection of paths $P_1, \ldots, P_k$ such that each $P_i$ is a shortest path from $s_i$ to $t_i$, and every node in the graph is on at most $c$ paths $P_i$, or reporting that no such collection of paths exists. When $c = k$, there are no congestion constraints, and the problem can be solved easily by running a shortest path algorithm for each pair $(s_i, t_i)$ independently. At the other extreme, when $c = 1$, the $(k, c)$-SPC problem is equivalent to the $k$-Disjoint Shortest Paths ($k$-DSP) problem, where the collection of shortest paths must be node-disjoint. For fixed $k$, $k$-DSP is polynomial-time solvable on DAGs and undirected graphs. Amiri and Wargalla interpolated between these two extreme values of $c$, to obtain an algorithm for $(k, c)$-SPC on DAGs that runs in polynomial time when $k - c$ is constant.

In the same way, we prove that $(k, c)$-SPC can be solved in polynomial time on undirected graphs whenever $k - c$ is constant. For directed graphs, because of our counterexample to the original theorem statement, our roundtrip local-to-global result does not imply such an algorithm $(k, c)$-SPC. Even without an algorithmic application, our proof for directed graphs may be of broader interest because it characterizes intriguing structural properties of shortest paths in directed graphs.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms

**Keywords and phrases** disjoint paths, shortest paths, congestion, parameterized complexity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.8

**Related Version** *Full Version*: https://arxiv.org/abs/2211.07042

**Funding** *Shyan Akmal*: Supported in part by NSF grant CCF-2129139.
*Nicole Wein*: Supported by a grant to DIMACS from the Simons Foundation (820931).

**Acknowledgements** We thank Virginia Vassilevska Williams for helpful discussions about this work.

## 1 Introduction

An intriguing question in graph theory and algorithms is: "can we understand the structure of shortest paths in (directed and undirected) graphs?" More specifically: "can we understand the structure of the *interactions* between shortest paths in graphs?" This question has been approached from various angles in the literature. For instance, Bodwin [12] characterizes which sets of nodes can be realized as *unique* shortest paths in weighted graphs, and Cizma

and Linial investigate the properties of graphs whose shortest paths satisfy or violate certain geometric properties [19, 20]. Additionally, there is a large body of work on distance preservers, where the goal is to construct a subgraph that preserves distances in the original graph (see [11] and the references therein). There are also numerous computational tasks in which structural results about shortest paths inform the creation and analysis of algorithms, including distance oracle construction [15], the $k$-disjoint shortest paths problem, [8], and the next-to-shortest path problem [35] (the cited papers are the most recent publications in their respective topics).

The angle of our work is inspired by the following local-to-global theorem[1] of Amiri and Wargalla [3] concerning the structure of shortest paths in directed acyclic graphs (DAGs):

▶ **Theorem 1** ([3]). *Let $G$ be a weighted DAG with the property that, for each set $S$ of 3 nodes, there is a shortest path containing every node in $S$. Then, there exists a pair $(s, t)$ of nodes such that there is a shortest $st$-path containing every node in $G$.*

Note that in the statement of Theorem 1, the condition that "for each set $S$ of 3 nodes, there is a shortest path containing every node in $S$" is equivalent to the condition that one of the 3 nodes is on a shortest path between the other two.

Theorem 1 is "local-to-global" in the sense that from a highly congested local structure (every small subset of nodes is contained in a shortest path) we deduce a global structure (all nodes in the graph live on a single shortest path).

At first glance, Theorem 1 may appear rather specialized, since the existence of shortest paths through *all* triples of nodes is a rather strong condition. However, Amiri and Wargalla [3] show that Theorem 1 has applications to the $k$-Shortest Paths with Congestion $c$ ($(k, c)$-SPC) on problem on DAGs: in this problem we are given a DAG, and are tasked with finding a collection of shortest paths between $k$ given source/sink pairs, such that each node in the graph is on at most $c$ of the paths. We discuss this application in detail in Section 1.2.

Amiri and Wargalla [3] raised the question of whether their results can be extended to general graphs, both undirected and directed. Our work answers this question.

## 1.1 Structural Results

We ask the following question:

*Is Theorem 1 true for general (undirected and directed) graphs?*

Our first result answers this question affirmatively for undirected graphs (with constant 4 instead of 3).

▶ **Theorem 2** (Undirected graphs). *Let $G$ be a weighted undirected graph with the property that, for each set $S$ of 4 nodes, there is a shortest path containing every node in $S$. Then, there exists a pair $(s, t)$ of nodes such that some shortest $st$-path contains every node in $G$.*

The constant 4 in the statement of Theorem 2 cannot be replaced with 3, as seen by considering an undirected cycle on four vertices.

Theorem 2 implies a faster algorithm for $(k, c)$-SPC on *undirected* graphs, answering an open question raised in [3, Section 4]. We discuss the background of this problem and our improvement in Section 1.2.

[1] This is slightly different from the statement of the theorem in [3], where the authors write present their result in the context of the Shortest Paths with Congestion problem, which we discuss in detail later.

Our second result is for directed graphs. First, we observe that there is actually a *counterexample* to Theorem 1 for general directed graphs: let $a$ be the constant for which we desire a counterexample (i.e., the constant that is 3 in Theorem 1, and 4 in Theorem 2). Let $G$ be a cycle with bidirectional edges, where all clockwise-pointing edges have weight 1 and all counterclockwise-pointing edges have weight $a$. One can verify that the precondition of the theorem holds: for each set $S$ of $a$ nodes there is a shortest path containing every node in $S$, just by taking the shortest clockwise path through all nodes in $S$. However, no single shortest path contains every node in $G$, so the theorem does not hold.

Even though a direct attempt at generalizing Theorem 1 to all directed graphs fails, one might hope for some analogue of Theorem 1 that does hold. One interpretation of the above counterexample is that the exact statement of Theorem 1 is not the "right" framework for getting a local-to-global shortest path phenomenon in general directed graphs. To that end, we consider the *roundtrip* analogue of Theorem 1, where the final path through every node is a shortest roundtrip path, i.e., the union of a shortest $st$-path and a shortest $ts$-path (roundtrip distances are a common object of study in directed graphs, with there being much research, for example, in roundtrip routing [21], roundtrip spanners [33], and roundtrip diameter computation [1]). Note that the above counterexample does not apply to the roundtrip analogue of Theorem 1 since there exists a pair $(s, t)$ of nodes such that the union of a shortest $st$-path and a shortest $ts$-path are both in the clockwise direction and thus contain all nodes in the graph.

For our second result, we present a roundtrip analogue of Theorem 1 which holds true for general directed graphs (with the constant 11 instead of 3):

▶ **Theorem 3** (Directed graphs). *Let $G$ be a weighted directed graph with the property that, for each set $S$ of 11 nodes, there is a shortest path containing every node in $S$. Then, there exists a pair $(s, t)$ of nodes such that the union of a shortest $st$-path and a shortest $ts$-path contains every node in $G$.*

Proving Theorem 3 requires overcoming a number of technical challenges involving the c omplex structure of shortest paths in directed graphs. Due to its roundtrip nature, unlike Theorem 2, Theorem 3 does not appear to have any immediate algorithmic applications.

## 1.2 Disjoint and Congested Shortest Path Problems

In this section we introduce the $k$-Shortest Paths with Congestion $c$ ($(k, c)$-SPC) problem and state the implications of our work for this problem.

### 1.2.1 Background

We begin by discussing the related $k$-Disjoint Shortest Paths ($k$-DSP) problem. For more related work on disjoint path problems in general, see the full version.

#### Disjoint Shortest Paths

Formally, the $k$-Disjoint Shortest Paths ($k$-DSP) problem is defined as follows:

> $k$-Disjoint Shortest Paths ($k$-DSP): Given a graph $G$ and $k$ node pairs $(s_1, t_1), \ldots, (s_k, t_k)$, find a collection of node-disjoint paths $P_1, \ldots, P_k$ such that each $P_i$ is a shortest path from $s_i$ to $t_i$, or report that no such collection of paths exists.

The $k$-DSP problem was introduced in the 90s by Eilam-Tzoreff [22], who gave a polynomial-time algorithm for undirected graphs when $k = 2$, and conjectured that there is a polynomial-time algorithm for any fixed $k$ in both undirected and directed graphs. Recently, Lochet [30] proved Eilam-Tzoreff's conjecture for undirected graphs by showing that $k$-DSP can be solved in polynomial time for any fixed $k$. Subsequently, the dependence on $k$ in the running time was improved by Bentert, Nichterlein, Renken, and Zschoche [8]. It is known that $k$-DSP on undirected graphs is W[1]-hard [8, Proposition 36], so this problem is unlikely to be fixed-parameter tractable.

For directed graphs, Bérczi and Kobayashi [9] showed that 2-DSP can be solved in polynomial time. For $k \geq 3$ however, determining the complexity of $k$-DSP on directed graphs remains a major open problem. This problem is only known to be polynomial-time solvable for special classes of directed graphs, such as DAGs and planar graphs [9].

### Shortest Paths with Congestion

The $k$-Shortest Paths with Congestion $c$ ($(k, c)$-SPC) problem is the variant of $k$-DSP where some amount of *congestion* (paths overlapping at nodes) is allowed. In general, problems of finding paths with limited congestion are well-studied studied in both theory and practice. For instance, there is much work on the problem in undirected graphs where the goal is to find a maximum cardinality subset of node pairs $(s_i, t_i)$ that admit (not necessarily shortest) paths with congestion at most $c$ [32, 28, 7, 6, 5, 17, 4, 26, 16, 18]. As another example, [27] provides, for fixed $k$, a polynomial-time algorithm for the problem on directed graphs of determining that either there is no set of disjoint paths between the node pairs $(s_i, t_i)$, or finding a set of such paths with congestion at most 4. Another example for directed graphs is the problem of finding paths between the node pairs $(s_i, t_i)$ where only some nodes in the graph have a congestion constraint [31].

Formally, the $k$-Shortest Paths with Congestion $c$ ($(k, c)$-SPC) problem is defined as follows:

> $k$-Shortest Paths with Congestion $c$ ($(k, c)$-SPC): Given a graph $G$, along with $k$ node pairs $(s_1, t_1), \ldots, (s_k, t_k)$, and a positive integer $c \leq k$, find a collection of paths $P_1, \ldots, P_k$ such that each $P_i$ is a shortest path from $s_i$ to $t_i$, and every node in $V(G)$ is on at most $c$ paths $P_i$, or report that no such collection of paths exists.

The $(k, c)$-SPC problem was introduced by Amiri and Wargalla [3]. Before that, the version of $(k, c)$-SPC where the paths are not restricted to be shortest paths was studied by Amiri, Kreutzer, Marx, and Rabinovich [2].

When $c = 1$, the $(k, c)$-SPC problem is equivalent to the $k$-DSP problem. At the other extreme, when $c = k$, there are no congestion constraints, so the problem can be easily solved in polynomial time by simply finding a shortest path for each pair $(s_i, t_i)$ independently. Amiri and Wargalla [3] asked the following question: *can we interpolate between these two extremes?* In particular, can we get algorithms for $(k, c)$-SPC where the exponential dependence on $k$ for $k$-DSP can be replaced with some dependence on $O(k - c)$ instead?

Amiri and Wargalla [3] achieved this goal for DAGs. In particular, they gave a reduction from $(k, c)$-SPC on DAGs to $k$-DSP on DAGs of the following form: letting $d = k - c$, if $k$-DSP on DAGs can be solved in time $f(n, k)$, then $(k, c)$-SPC on DAGs can be solved in time $O\left(\binom{k}{3d} \cdot f(2dn, 3d)\right)$. The essential aspect of this running time is that the second input to the function $f$ is not $k$ but rather an $O(d)$ term. A key tool in their reduction is Theorem 1 (stated in a different way).

Since $k$-DSP can be solved in $n^{O(k)}$ time on DAGs [9], the above result implies that $(k, c)$-SPC can be solved in $\binom{k}{3d} \cdot (2dn)^{O(d)}$ time on DAGs. That is, $(k, c)$-SPC on DAGs is polynomial-time solvable for arbitrary $k$ whenever $d$ is constant. We note that for for every $c$, the $(k, c)$-SPC problem on DAGs is W[1]-hard with respect to $d$, so the problem is unlikely to be fixed-parameter tractable with respect to $d$ [3, Proof of Theorem 3].

### 1.2.2 Algorithmic Results

Similarly to Amiri and Wargalla's result for DAGs, our result for undirected graphs, Theorem 2, implies a reduction from $(k, c)$-SPC to $k$-DSP on undirected graphs.

▶ **Lemma 4.** *If $k$-DSP can be solved in $f(n, k)$ time on undirected graphs, then $(k, c)$-SPC can be solved in $O\left(\binom{k}{4d} \cdot f(3dn, 4d)\right)$ time on undirected graphs.*

Lemma 4 follows from Theorem 2 using an argument nearly identical to the one presented for DAGs in [3] (up to a difference in constants). For completeness, we include a full proof of this result in the full version.

Since it is known that $k$-DSP can be solved in undirected graphs in time $n^{O(k \cdot k!)}$ [8], applying Theorem 2 together with Lemma 4, we deduce the following result.

▶ **Corollary 5.** *$(k, c)$-SPC on undirected graphs can be solved in $\binom{k}{4d} \cdot (3dn)^{O(d \cdot (4d)!)}$ time.*

Thus $(k, c)$-SPC on undirected graphs is in polynomial time whenever $d = k - c$ is constant; that is, it is in the complexity class XP with respect to the parameter $d$. Prior to our work, no polynomial-time algorithm for this problem appears to have been known in this regime, even for simple cases such as $(k, k-1)$-SPC on undirected graphs.

In contrast, our structural result for directed graphs, Theorem 3, does not imply a faster algorithm for $(k, c)$-SPC in directed graphs. This is because Theorem 3 does not appear to imply a reduction from $(k, c)$-SPC to $k$-DSP in the manner of Lemma 4. Moreover, even if such a reduction did exist, this would not imply an algorithm for directed graphs analogous to Corollary 5. This is because while $k$-DSP is polynomial-time solvable for constant $k$ in undirected graphs, it remains open whether even 3-DSP over directed graphs can be solved in polynomial time.

## 1.3 The Structure of Shortest Paths in Directed Graphs

Although our result for directed graphs, Theorem 3, does not appear to have immediate algorithmic applications, we believe it is still interesting from a graph theoretic perspective, especially in light of the current scarcity of results for shortest disjoint path problems in directed graphs. In this section, we expand upon this idea with some remarks, and then state a lemma from our proof concerning the structure of shortest paths in directed graphs.

We currently have a poor understanding of the complexity of the $k$-DSP problem in directed graphs: for fixed $k \geq 3$, it is still not known if this problem is either polynomial-time solvable or NP-hard. In fact, even the complexity of the seemingly easier $(3, 2)$-SPC problem on directed graphs is open. In this context, Theorem 3 is compelling because it presents an example of interesting behavior which holds for collections of shortest paths in DAGs, and then continues to hold, under suitable generalization, for systems of shortest paths in general directed graphs. This sort of characterization appears to be rare in the literature.

More generally, the methods we use to establish Theorem 3 involve combinatorial observations about the structure of shortest paths in directed graphs, and the interactions between them. We believe our analysis could offer more insight into resolving other problems that

concern systems of shortest paths in directed graphs. There are many such problems, where the undirected case is well-understood, but in the directed case not much is known. This barrier is in-part due to the relatively complex patterns of shortest paths which can appear in directed graphs. We hope that our analysis of directed shortest paths may shed light on problems for which the structural complexity of directed shortest paths is the bottleneck towards progress.

One example of a problem where the undirected case is well-understood while the directed case remains poorly understood is, as discussed previously, the $k$-DSP problem. Another curious example is the Not-Shortest Path problem, where the goal is to find an $st$-path that is not a shortest path. Although Not-Shortest Path can be solved over undirected graphs in polynomial time [29], no polynomial time algorithm is known for this problem in directed graphs. A third example is the problem of approximating the diameter of a graph. For undirected graphs there is an infinite hierarchy of algorithms that trade off between time and accuracy [14], while only two points on the hierarchy are known for directed graphs. Additionally, the roundtrip variant of diameter is the least understood of any studied variant of the diameter problem [34]. Another example is the construction of approximate hopsets: for directed graphs there is there a polynomial gap between upper and lower bounds [25, 10, 13], while for undirected graphs the gap is subpolynomial [23, 24]. The preponderance of such examples motivates proving results like Theorem 3, which characterize interesting behavior of shortest paths in directed graphs.

### Structural Lemma for Directed Shortest Paths

One of the lemmas we establish on the way to proving Theorem 3 is a general statement about the structure of shortest paths in directed graphs. It can be stated independently of the context of the proof of Theorem 3 and we highlight it here.

We categorize any shortest path $P$ into one of two main types, based on the ways that other shortest paths intersect with it. The following simple definition will be useful for defining our path types.

▶ **Definition 6.** *For any nonnegative integer $\ell$ and set of $\ell$ nodes, $v_1, v_2, \ldots, v_\ell$, we say that the order $v_1 \to v_2 \to \cdots \to v_\ell$ is a* shortest-path ordering *if there is a shortest path containing all of the nodes $v_1, v_2, \ldots, v_\ell$ in that order.*

In addition to our two main path types, there is a third path type which we call a *trivial path* because it is easy to handle:

▶ **Definition 7** (Trivial Path)**.** *Given a directed graph, nodes $a$ and $b$, and a shortest path $P$ from $a$ to $b$, we say $P$ is a* trivial path *if $P$ contains at least one node $w$ such that $a \to w \to b$ is the* only *shortest-path ordering of $a, w, b$.*

Now we are ready to state our two main types of shortest paths. The first type is a *reversing path*:

▶ **Definition 8** (Reversing path)**.** *Given a directed graph, nodes $a$ and $b$, and a non-trivial shortest path $P$ from $a$ to $b$, $P$ is* reversing *if $P$ contains at least one node $w$ such that $w$ falls on some shortest path from $b$ to $a$. A* non-reversing path *is a non-trivial path that is not reversing.*

We prove a lemma that characterizes the structure of reversing and non-reversing paths in terms of the possible shortest-path orderings of each node on the path and the endpoints of the path. See Figures 1a and 1b for a depiction of the structure enforced by the lemma.

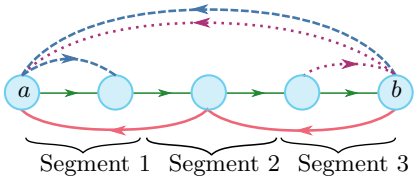
▶ **Lemma 9** (Reversing/Non-Reversing Lemma). *Let $P$ be a non-trivial shortest path and let $a$ and $b$ be the first and last nodes of $P$ respectively. Then $P$ can be partitioned into three contiguous ordered segments with the following properties (where $a$ is defined to be in Segment 1, $b$ is defined to be in Segment 3, and Segment 2 could be empty).*

*Segment 1 consists of nodes $w$ such that the shortest-path orderings of $a, w, b$ are precisely $a \to w \to b$, and $b \to a \to w$.*

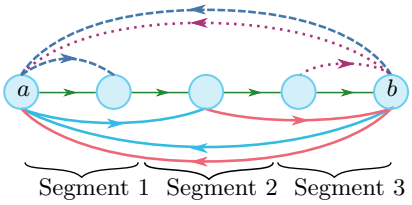
*Segment 2 consists of nodes $w$ such that the shortest-path orderings of $a, w, b$ are precisely*

$$\begin{cases} a \to w \to b, \text{ and } b \to w \to a & \text{if } P \text{ is a reversing path} \\ a \to w \to b, \text{ and } b \to a \to w, \text{ and } w \to b \to a & \text{if } P \text{ is a non-reversing path.} \end{cases}$$

*Segment 3 consists nodes $w$ such that the shortest-path orderings of $a, w, b$ are precisely $a \to w \to b$, and $w \to b \to a$.*




**(a) Reversing Path.** The structure of a reversing path, as given by Lemma 9. The blue dashed path, pink solid path, and purple dotted path are examples of the allowed orderings for nodes in segments 1, 2, and 3 respectively.




**(b) Non-Reversing Path.** The structure of a non-reversing path, as given by Lemma 9. The possible shortest-path orderings for nodes in segment 2 are represented by pink and light blue solid paths, while the orderings allowed for nodes in segments 1 and 3 are represented by blue dashed and purple dotted segments respectively.

**Figure 1** Possible orderings of vertices on shortest paths in the reversing and non-reversing cases. The blue circles are representative examples of the types of nodes on the path from $a$ to $b$ (in general this path will contain more than just five nodes).

In the proof of Theorem 3 we employ the strategy of categorizing shortest paths as reversing or non-reversing (or trivial), and applying Lemma 9 to glean some structure. We note, however, that Lemma 9 itself is not the main technical piece of the proof.

## 2 Preliminaries

All graphs are assumed to have positive edge weights. Graphs are either undirected or directed, depending on the section. For any pair of nodes $(u, v)$, we use $\mathsf{dist}(u, v)$ to denote the shortest path distance from $u$ to $v$. Given a path $P$ and two nodes $u$ and $v$ occurring on $P$ in that order, we let $P[u, v]$ denote the subpath of $P$ with $u$ and $v$ as endpoints.

For the $(k, c)$-SPC problem, we always let $d$ denote the difference $d = k - c$. When considering a particular solution to a $(k, c)$-SPC instance, we refer to the paths $P_1, \ldots, P_k$ between $(s_1, t_1), \ldots, (s_k, t_k)$ respectively as *solution paths*. Any node in the graph which lies in $c$ of the solution paths is referred to as a *max-congestion* node.

## 2.1 Subpath Swapping

In our proofs, we will frequently modify collections of shortest paths by "swapping subpaths" between intersecting paths. This procedure is depicted in Figure 2, and we formally describe it below.

▶ **Definition 10** (Subpath Swap). *Let $\mathcal{R}$ be a collection of shortest paths in a directed graph. Let $P$ and $Q$ be two paths in $\mathcal{R}$. Let $a, b \in P \cap Q$ be nodes in these paths, such that $a$ appears before $b$ in both $P$ and $Q$. We define* swapping *the subpaths of $P$ and $Q$ between $a$ and $b$ to be updating the set of paths $\mathcal{R}$ by simultaneously replacing $P$ with $(P \setminus P[a,b]) \cup Q[a,b]$ and $Q$ with $(Q \setminus Q[a,b]) \cup P[a,b]$. We often refer to this process simply as "swapping $P[a,b]$ and $Q[a,b]$."*


**Figure 2** A simple subpath swap, where the subpaths from $a$ to $b$ of the green path (from $s_i$ to $t_i$) and pink path (from $s_j$ to $t_j$) are switched.

▶ **Observation 11** (Subpath Swap). *Let $\mathcal{R}$ be the solution to some $(k,c)$-SPC problem. Then swapping subpaths in $\mathcal{R}$ produces a new solution to the same $(k,c)$-SPC instance with the same set of max-congestion nodes.*

**Proof.** This observation holds because swapping subpaths does not change the number of solution paths any given node is contained in, does not change the endpoints of any solution path, and all solution paths remain shortest paths. ◀

## 2.2 Correspondence Between Our Results and $(k, c)$-SPC

In this section we detail some nuances regarding the correspondence between the statement of our results and the $(k,c)$-SPC problem. For the sake of generality and simplicity, we stated Theorems 1-3 independently of the $(k,c)$-SPC problem. In contrast, the original result of Amiri and Wargalla, corresponding to Theorem 1, was stated as follows:

▶ **Lemma 12** ([3]). *If $k > 3d$, then any instance of $(k,c)$-SPC on DAGs either has no solution, or has a solution where some solution path $P_i$ passes through all max-congestion nodes.*

Although the statement of Lemma 12 may initially seem unrelated to the statement of Theorem 1, their correspondence becomes clearer with the following observation, which is a simple generalization of an observation from [3]:

▶ **Observation 13.** *Let $N$ be a positive integer. Suppose $k > Nd$, and let $\mathcal{R}$ be a solution to an arbitrary $(k,c)$-SPC instance. Then for any set $S$ of $N$ max-congestion nodes in $\mathcal{R}$, there exists some solution path in $\mathcal{R}$ which contains every node in $S$.*

We defer the proof of Observation 13 to the appendix.

To prove our results for the $(k,c)$-SPC problem in undirected graphs (Lemma 4 and Corollary 5), we need to prove a lemma analogous to Lemma 12 but for undirected graphs:

▶ **Lemma 14.** *If $k > 4d$, then any instance of $(k, c)$-SPC either has no solution, or has a solution where some solution path $P_i$ passes through all max-congestion nodes.*

The following generalizes Theorem 2 and Lemma 14.

▶ **Lemma 15** (General Undirected Result)**.** *Given an undirected graph and subset $W$ of nodes, let $\mathcal{R}$ be a collection of shortest paths with the following property:*

$$\text{for every set } S \subseteq W \text{ of 4 nodes, some path in } \mathcal{R} \text{ contains every node in } S. \qquad (\star)$$

*Further suppose that applying any sequence of $O(n^3)$ subpath swaps to $\mathcal{R}$ yields a collection of shortest paths that still has property $(\star)$. Then starting from $\mathcal{R}$, there exists a sequence of subpath swaps that results in a collection of shortest paths in which some path $P$ passes through all nodes in $W$.*

We note that the quantity $O(n^3)$ is an unimportant technicality used in the proof of the following observation, and is chosen as a loose upper bound on the number of subpath swaps we will perform.

▶ **Observation 16.** *Lemma 15 generalizes both Theorem 2 and Lemma 14.*

We defer the proof of Observation 16 to the appendix.

We also prove an analogue of Lemma 12 and Lemma 15 for directed graphs:

▶ **Lemma 17** (General Directed Result)**.** *Given a directed graph and subset $W$ of nodes, let $\mathcal{R}$ be a collection of shortest paths with the following property:*

$$\text{for every set } S \subseteq W \text{ of 11 nodes, some path in } \mathcal{R} \text{ contains every node in } S. \qquad (\dagger)$$

*Further suppose that applying any sequence of $O(n^3)$ subpath swaps to $\mathcal{R}$ yields a collection of paths that still has property $(\dagger)$. Then starting from $\mathcal{R}$, there exists a sequence of subpath swaps that results in a collection of shortest paths in which either:*

1. *some path $P$ passes through all nodes in $W$, or*
2. *the union of two paths $P, P'$ contain all nodes in $W$, and the first and last nodes on $P$ from $W$ are the same as the last and first nodes on $P'$ from $W$, respectively.*

Lemma 17 generalizes Theorem 3, in exactly the same way as Lemma 15 generalizes Theorem 2 for undirected graphs. In the same way that Lemma 15 generalizes Lemma 14 for undirected graphs, Lemma 17 implies the following lemma:

▶ **Lemma 18.** *If $k > 11d$, then any instance of $(k, c)$-SPC on directed graphs either has no solution, or has a solution where the union of some two solution paths $P_i$ and $P_j$ contains all max-congestion nodes.*

Although Lemma 18 does not immediately lead to an algorithm for $(k, c)$-SPC on directed graphs, it specifies some structure which may be useful for future work on $(k, c)$-SPC and related problems.

One might wonder whether Lemma 18 can be modified to have only one solution path $P_i$ that contains all max-congestion nodes, like for DAGs (Lemma 12) and undirected graphs (Lemma 14), since this would imply interesting algorithms for $(k, c)$-SPC on directed graphs. Unfortunately, the answer to this question turns out to be no. Similar to the counterexample against extending Theorem 2 to directed graphs, we present a counterexample in the full version which rules out replacing two solution paths with a single solution path in Lemma 18.

# 3 Technical Overview

## 3.1 Prior Work on DAGs

As a starting point for our proofs for general graphs, we use Amiri and Wargalla's proof of Theorem 1 for DAGs [3]. We describe their proof differently than they do for the sake of comparison to our work. We actually describe their proof of the following lemma (which they do not explicitly state), which is analogous to the statements of our general results (Lemmas 15 and 17).

▶ **Lemma 19** ([3]). *Given a DAG and subset $W$ of nodes, let $\mathcal{R}$ be a collection of shortest paths with the following property:*

$$\text{for every set } S \subseteq W \text{ of 3 nodes, some path in } \mathcal{R} \text{ contains every node in } S. \qquad (*)$$

*Further suppose that applying any sequence of $O(n^3)$ subpath swaps to $\mathcal{R}$ yields a collection of paths that still has property $(*)$. Then starting from $\mathcal{R}$, there exists a sequence of subpath swaps that results in a collection of shortest paths in which some path $P$ passes through all nodes in $W$.*

The proof of Lemma 19 is as follows. Let $a$ and $b$ be the first and last nodes in $W$ (respectively) in a topological ordering of the DAG. Let $v_1, \dots v_{|W|-2}$ be the remaining nodes in $W$ in order topologically. For ease of notation, we consider $\mathcal{R}$ to be changing over time via subpath swaps, and we will let $\mathcal{R}$ denote the current value of $\mathcal{R}$.

The argument is inductive. For the base case, by property $(*)$ there is a path in $\mathcal{R}$ that contains $a$ and $b$. Suppose inductively that a path $P \in \mathcal{R}$ currently contains $a$, $b$, and $v_1, \dots, v_\ell$ for some $\ell$. We would like to perform a subpath swap to augment $P$ by adding $v_{\ell+1}$ to $P$. To do so, we consider a path $P' \in \mathcal{R}$ that contains $v_\ell$, $v_{\ell+1}$, and $b$, where such a path exists by property $(*)$. Importantly, because the graph is a DAG, $v_\ell$, $v_{\ell+1}$, and $b$ appear in that order on both $P$ and $P'$. Thus, according to the definition of a subpath swap (Definition 10) we can swap $P[v_\ell, b]$ with $P'[v_\ell, b]$, as shown in Figure 3. As a result of this subpath swap, $P$ now contains $v_{\ell+1}$ in addition to all of the nodes in $W$ that $P$ originally contained. By induction, this completes the proof.

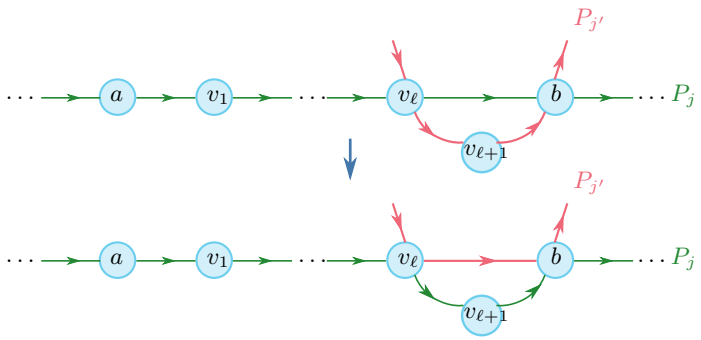



**Figure 3** In a DAG, the topological ordering of the nodes allows us to perform a sequence of subpath swaps, each adding the next node in $W$ in order to a path $P$.

## 3.2 Undirected Graphs

Recall that our goal for undirected graphs is to prove the following theorem:

▶ **Lemma 15** (General Undirected Result). *Given an undirected graph and subset $W$ of nodes, let $\mathcal{R}$ be a collection of shortest paths with the following property:*

$$\text{for every set } S \subseteq W \text{ of 4 nodes, some path in } \mathcal{R} \text{ contains every node in } S. \qquad (\star)$$

*Further suppose that applying any sequence of $O(n^3)$ subpath swaps to $\mathcal{R}$ yields a collection of shortest paths that still has property $(\star)$. Then starting from $\mathcal{R}$, there exists a sequence of subpath swaps that results in a collection of shortest paths in which some path $P$ passes through all nodes in $W$.*

The essential property that enables the subpath swapping in the above argument for DAGs is the fact that for any triple of nodes in a DAG, there is *only one* possible order this triple can appear on any path. This property is not exactly true for general undirected graphs, but we observe that a similar "consistent ordering" property is true: if there is a *shortest* path containing nodes $u, v, w$ in that order, then any shortest path containing these nodes, has them in that order (or in the reverse order $w, v, u$, but since the graph is undirected we can without loss of generality assume they are in the order $u, v, w$). This property is true simply because $\mathsf{dist}(u, w)$ is larger than both $\mathsf{dist}(u, v)$ and $\mathsf{dist}(v, w)$, so $v$ must appear between $u$ and $w$ on any shortest path.

To perform subpath swapping on undirected graphs, we need an initial pair of nodes in $W$ such that the rest of the nodes in $W$ will be inserted between this initial pair (in the DAG algorithm, this initial pair $a, b$ was the first and last nodes in $W$ in the topological order). For undirected graphs, our initial pair is the pair $a, b$ of nodes in $W$ whose distance is maximum. We order the rest of the nodes in $W$ by their distance from $a$, to form $v_1, \ldots, v_{|W|-2}$.

Now, our consistent ordering property from above implies the following: for any shortest path $P$ containing $a$, all nodes in $W \cap P$ are ordered as a subsequence of $a, v_1, v_2, \ldots, v_m, b$ on $P$. As a result, we can perform the same type of iterative subpath swapping argument as the DAG algorithm.

## 3.3 Roundtrip Paths in Directed Graphs

The situation for directed graphs is significantly more involved than the previous cases. There are several challenges that are present for directed graphs that were not present for either undirected graphs or DAGs. These difficulties stem from the fact that the *interactions* between shortest paths is much more complicated in directed graphs than in DAGs or undirected graphs.

We first outline these challenges, and then provide an overview of how we address them. Our techniques for addressing these issues exemplify that despite the possibly complex arrangement of shortest paths in directed graphs, there still exists an underlying structure to extract. We hope that our methods might illuminate some structural properties of shortest paths in directed graphs in a way that could apply to other directed-shortest-path problems.

Recall that our goal is to prove the following theorem:

▶ **Lemma 17** (General Directed Result). *Given a directed graph and subset $W$ of nodes, let $\mathcal{R}$ be a collection of shortest paths with the following property:*

$$\text{for every set } S \subseteq W \text{ of 11 nodes, some path in } \mathcal{R} \text{ contains every node in } S. \qquad (\dagger)$$

*Further suppose that applying any sequence of $O(n^3)$ subpath swaps to $\mathcal{R}$ yields a collection of paths that still has property (†). Then starting from $\mathcal{R}$, there exists a sequence of subpath swaps that results in a collection of shortest paths in which either:*

1. *some path $P$ passes through all nodes in $W$, or*
2. *the union of two paths $P, P'$ contain all nodes in $W$, and the first and last nodes on $P$ from $W$ are the same as the last and first nodes on $P'$ from $W$, respectively.*

## Challenges for directed graphs

### Challenge 1: No "extremal" nodes

In the proofs for DAGs and undirected graphs, to begin building the path $P$ containing all nodes in $W$, we chose two initial extremal nodes $a, b \in W$ and added the rest of the nodes of $W$ between $a$ and $b$. These nodes $a, b$ were straightforward to choose because there was only one pair of nodes in $W$ that could possibly appear first and last on a path containing all nodes in $W$ (for DAGs $a$ and $b$ were the first and last nodes in a topological ordering of $W$, and for undirected graphs $a$ and $b$ were the pair of nodes in $W$ with largest distance).

For directed graphs however, it is entirely unclear how to pick these two initial extremal nodes. For instance, choosing the pair of nodes $a, b \in W$ with largest directed distance $\mathsf{dist}(a, b)$ does not work because there could be a shortest path $Q$ containing $a$ and $b$, where $a$ and $b$ are not the first and last nodes in $W$ on $Q$ (in particular, if $b$ appears before $a$ on the shortest path).

To circumvent this issue for directed graphs, we avoid selecting a pair of initial nodes at all. Without initial nodes as an anchor, we cannot build our path $P$ in order from beginning to end as we did for DAGs and undirected graphs. Instead, our goal is to ensure the following weaker property: as we iteratively transform the overall collection of shortest paths $\mathcal{R}$, the set of nodes in $W$ on the path in $\mathcal{R}$ containing the most nodes in $W$ grows over time. That is, the path of $\mathcal{R}$ containing the most nodes in $W$ might currently be $P$, but at the previous iteration, the path of $\mathcal{R}$ with the most nodes in $W$ might have been a different path $P'$. In this case, our weaker property ensures that $P$ currently contains a *superset* of the nodes in $W$ that $P'$ contained at the previous iteration.

To perform a single iteration with this guarantee, we may need to significantly reconfigure *many* different paths of $\mathcal{R}$ via many subpath swaps. As a result, our path building procedure is much more intricate than the procedures employed for DAGs and undirected graphs.

### Challenge 2: No consistent ordering of nodes on shortest paths

In the proof for DAGs and undirected graphs, we were able to perform subpath swaps due to the following crucial property: consider an arbitrary set of nodes $v_1, \ldots, v_\ell$ (for any $\ell$) in a DAG or an undirected graph. If there is a shortest path containing the nodes $v_1, \ldots, v_\ell$ in that order, then *every* shortest path containing these nodes has them in that same order.

For directed graphs, however, this property is not even close to being true. In fact, given that the nodes $v_1, \ldots, v_\ell$ appear in that order on some shortest path, there are *exponentially many* other possible orderings of these nodes on other shortest paths. For instance, when $\ell = 4$, given that the nodes $v_1, v_2, v_3, v_4$ appear in that order on some shortest path, there are eight possible orderings of these nodes on shortest paths, as depicted in Figure 4 (note that despite the large number of possible orderings, not all orderings are possible; for instance the ordering $v_1, v_2, v_4, v_3$ is not possible, as this would imply that $\mathsf{dist}(v_1, v_4) < \mathsf{dist}(v_1, v_3)$, which contradicts our assumption that some shortest path contains $v_1, v_2, v_3, v_4$ in that order).

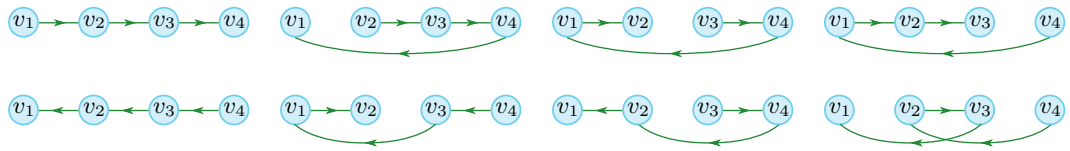



**Figure 4** If nodes $v_1, v_2, v_3, v_4$ appear in that order on some shortest path, they can still appear on different shortest paths in up to seven other distinct possible orders.

Because directed graphs have no consistent ordering of nodes on shortest paths, it becomes much more difficult to perform subpath swaps like those in the algorithms for DAGs and undirected graphs.

To address this challenge, we provide a structural analysis of the ways in which shortest paths in directed graphs can interact with each other. First, as introduced in Section 1.3, we categorize shortest paths into two main types, *reversing* paths and *non-reversing* paths, and we prove the Reversing/Non-Reversing Path Lemma (Lemma 9). Roughly speaking, this lemma is useful because it helps us construct sets of nodes that exhibit some sort of consistent ordering property. This, in turn, enables us to perform sequences of subpath swaps. Defining these consistently ordered sets of nodes and the corresponding subpath swaps is the most involved part of the proof, and works differently for each of the two path types.

## Proof structure

Our proof is structured as follows. Initially, we define $P$ to be the path in $\mathcal{R}$ that contains the most nodes in $W$ (breaking ties arbitrarily). Then we proceed with the following two cases:

- (Case 1) We first check whether, roughly speaking, $P$ is contained in a *cycle* that contains *all* nodes in $W$. In this case, we can use a sequence of subpath swaps to build a second path $P'$ so that the union of $P$ and $P'$ contains all nodes in $W$ and have the "roundtrip" structure specified in the theorem statement, in which case we are done.
- (Case 2) If we are not in Case 1, our goal is to augment some path in $\mathcal{R}$ so that the set of nodes of $W$ on the path in $\mathcal{R}$ with the most nodes in $W$ grows (the goal introduced in the discussion of Challenge 1).

After going through these cases, if we are not done we redefine $P$ as the path in $\mathcal{R}$ containing the most nodes in $W$ and repeatedly apply the appropriate case, until we are done. Most technical details of our proof are in the path augmentation procedure of Case 2. We elaborate on the main ideas for this procedure next.

## Handling Case 2: Path Augmentation

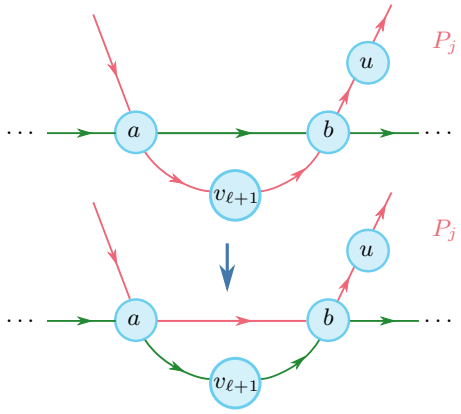
Recall that our goal is the following: Let $P$ be the path in $\mathcal{R}$ containing the most nodes in $W$, and let $W' = W \cap P$. Fix a node $u \in W \setminus W'$. We wish to perform subpath swaps to yield a path $P'$ that contains every node in $W' \cup \{u\}$.

We begin with a few warm-up cases to motivate our general approach.

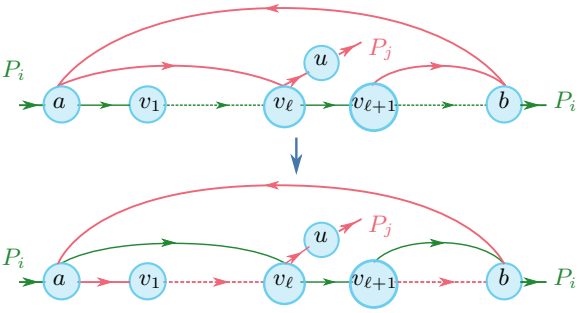
### Warm-up cases

Let $a$ and $b$ be the first and last nodes on $P$ that are in $W$, respectively. By property (†) there exists a path in $\mathcal{R}$ containing $a$, $b$, and $u$. We will suppose in all of the subsequent examples, that for *all* paths in $\mathcal{R}$ containing the nodes $a$, $b$, and $u$, the node $u$ is always the last node in $W$ on the path; this is not a conceptually important assumption and it makes the description simpler.

We claim that if there is a path $P' \in \mathcal{R}$ containing $a$, $b$, and $u$, such that $a$ appears before $b$, then we are done. Indeed, as depicted in Figure 5a, in this case we can simply swap the subpath $P[a, b]$ with $P'[a, b]$, to form a new solution where $P'$ contains $W' \cup \{u\}$.




**(a)** If $a$ appears before $b$ on $P'$, a subpath swap lets $P'$ pass through $u$ together with all of the nodes in $W'$.




**(b)** More generally, when $P'$ passes through $v_{\ell+1}, b, a, v_\ell$ in that order, we can perform two subpath swaps to get $P'$ to pass through the rest of $W'$. Here, the dotted segments indicate portions of the paths which pass through the nodes of $W'$ that are not labeled in the figure.

**Figure 5** The two warm-up cases.

Now we will slightly generalize this warm-up case. Let $a, v_1, v_2, \ldots, b$ be the nodes in $W'$ in the order they appear on $P$. Consider $v_\ell$ and $v_{\ell+1}$ for any $\ell$. By property (†) there exists a path in $\mathcal{R}$ containing $a$, $b$, $u$, $v_\ell$, and $v_{\ell+1}$. We know from the previous warm-up case that if there is a pathin $\mathcal{R}$ containing these nodes such that $a$ appears before $b$, then we are done. We also claim that if there exists a path $P' \in \mathcal{R}$ containing these nodes such that $v_{\ell+1}, b, a, v_\ell$ appear in that order, then we are done. This is because as depicted in Figure 5b, we can swap the subpath $P[a, v_\ell]$ with $P'[a, v_\ell]$, and swap the subpath $P[v_{\ell+1}, b]$ with $P'[v_{\ell+1}, b]$. Now, $P'$ contains $W' \cup \{u\}$.

### General Approach: "Critical nodes"

We will motivate our general approach in the context of the above warm-up cases. In the second warm-up case, we considered 5 nodes ($a$, $b$, $u$, $v_\ell$, and $v_{\ell+1}$) in $W'$, and argued that if there is a path $P' \in \mathcal{R}$ containing these 5 nodes in one of several "good" orders, then we are done because we can perform subpath swaps to reroute $P'$ through all of $W' \cup \{u\}$. Thus, our goal is to choose these 5 (or in general, at most 11) nodes carefully, to guarantee that they indeed fall into a "good" order on some path in $\mathcal{R}$. We refer to these at most 11 nodes as *critical nodes*:

▶ **Definition 20** (Critical Nodes (Informal)). *Given a path $P \in \mathcal{R}$, a set of nodes $T \subseteq W$ of size $|T| \leq 11$ are* critical nodes of $P$ *if there exists a path $P' \in \mathcal{R}$ containing the nodes of $T$ in an order that allows us to perform subpath swaps to reroute some path in $\mathcal{R}$ through all of $W' \cup \{u\}$ (where $W' = W \cap P$).*

*Our general approach is to show that any path $P \in \mathcal{R}$ contains a set of at most 11 critical nodes.* We remind the reader that this section only concerns Case 2, so our goal is to show that $P$ contains a set of critical nodes only if $P$ does not already fall into Case 1. After showing that $P$ contains a set of critical nodes, we are done, because performing subpath swaps to yield a path containing $W' \cup \{u\}$ was our stated goal.

It is not at all clear a priori that any $P$ should contain a set of critical nodes. Indeed, the critical nodes need to be chosen very carefully. Specifically, they need to be chosen based on the structure of the path $P$.

This is where the definitions from Section 1.3 come into play. We categorize $P$ based on whether it is *reversing* or *non-reversing*, (or trivial). Then we construct the critical nodes of $P$ using a different procedure specialized for which type of path $P$ is.

If $P$ is a reversing path, then we argue that a valid choice of critical nodes are $a$, $b$, and $u$, along with the two nodes at the two boundaries between the segments defined in Lemma 9 (and a few other nodes for technical reasons). To make this argument, which we will not detail here, we construct an involved series of subpath swaps to reroute some path through all of $W' \cup \{u\}$.

On the other hand, if $P$ is a non-reversing path, the critical nodes are less straightforward to define than if $P$ is a reversing path. Indeed, defining the critical nodes for non-reversing paths is the most conceptually difficult part of our proof. The high-level reason for this difficulty is the fact that the nodes of Segment 2 of a non-reversing path are quite unconstrained because they admit 3 possible shortest-path orderings instead of only 2.

To be more concrete, suppose every node on $P$ falls into Segment 2. For simplicity, suppose we were to choose critical nodes $a, b, w$, where $a$ and $b$ are the endpoints of $P$ and also happen to be in $W'$, and $w$ is any other node in $W'$. Consider the path $P' \in \mathcal{R}$ containing $a$, $b$, and $w$, which exists by property (†). By the definition of Segment 2, there are 3 possible orderings of $a, b, w$ on $P'$ ($a \to w \to b$, or $b \to a \to w$, or $w \to b \to a$). Note that these 3 orderings are cyclic shifts of one another. Suppose, as an illustrative example, that $P'$ has the ordering $b \to a \to w$. We would like to reroute $P'$ through all of $W' \cup \{u\}$, but we have a problem. For any node $s \in W'$ that appears between $a$ and $w$ on $P$, we can reroute $P'$ through $s$ by swapping the subpath $P[a, w]$ with $P'[a, w]$; however, for a node $s \in W'$ that appears between $w$ and $b$ on $P$, we cannot do this because there is no path segment from $w$ to $b$ on $P'$, since $b$ appears before $w$ on $P'$. Thus, we cannot reroute $P'$ through $s$. That is, no matter how we choose the critical nodes, if we do not impose extra structural constraints, we can always identify a segment of the path that we cannot reroute $P'$ through. Overcoming this issue is our main technical challenge, and we defer it to the full proof.

## References


1. Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.
2. Saeed Akhoondian Amiri, Stephan Kreutzer, Dániel Marx, and Roman Rabinovich. Routing with congestion in acyclic digraphs. *Inf. Process. Lett.*, 151, 2019. doi:10.1016/j.ipl.2019.105836.
3. Saeed Akhoondian Amiri and Julian Wargalla. Disjoint shortest paths with congestion on dags. *CoRR*, abs/2008.08368, 2020. arXiv:2008.08368.

4 Matthew Andrews. Approximation algorithms for the edge-disjoint paths problem via raecke decompositions. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 277–286. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.33.

5 Matthew Andrews and Lisa Zhang. Hardness of the undirected congestion minimization problem. *SIAM J. Comput.*, 37(1):112–131, 2007. doi:10.1137/050636899.

6 Yossi Azar and Oded Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44(1):49–66, 2006. doi:10.1007/s00453-005-1172-z.

7 Alok Baveja and Aravind Srinivasan. Approximation algorithms for disjoint paths and related routing and packing problems. *Math. Oper. Res.*, 25(2):255–280, 2000. doi:10.1287/moor.25.2.255.12228.

8 Matthias Bentert, André Nichterlein, Malte Renken, and Philipp Zschoche. Using a Geometric Lens to Find k Disjoint Shortest Paths. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.26.

9 Kristóf Bérczi and Yusuke Kobayashi. The directed disjoint shortest paths problem. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 13:1–13:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.13.

10 Aaron Bernstein and Nicole Wein. Closing the gap between directed hopsets and shortcut sets, 2022. doi:10.48550/arXiv.2207.04507.

11 Greg Bodwin. Linear size distance preservers. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 600–615. SIAM, 2017. doi:10.1137/1.9781611974782.39.

12 Greg Bodwin. On the structure of unique shortest paths in graphs. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2071–2089. SIAM, 2019. doi:10.1137/1.9781611975482.125.

13 Greg Bodwin and Gary Hoppenworth. Folklore sampling is optimal for exact hopsets: Confirming the $\sqrt{n}$ barrier, 2023. doi:10.48550/arXiv.2304.02193.

14 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376. SIAM, 2016. doi:10.1137/1.9781611974331.ch27.

15 Shiri Chechik and Tianyi Zhang. Nearly 2-approximate distance oracles in subquadratic time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 551–580. SIAM, 2022. doi:10.1137/1.9781611977073.26.

16 Chandra Chekuri and Alina Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 326–341. SIAM, 2013. doi:10.1137/1.9781611973105.24.

17 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Edge-disjoint paths in planar graphs with constant congestion. *SIAM J. Comput.*, 39(1):281–301, 2009. doi:10.1137/060674442.

18 Julia Chuzhoy. Routing in undirected graphs with constant congestion. *SIAM J. Comput.*, 45(4):1490–1532, 2016. doi:10.1137/130910464.

19 Daniel Cizma and Nati Linial. Geodesic geometry on graphs. *Discrete & Computational Geometry*, 68(1):298–347, January 2022. doi:10.1007/s00454-021-00345-w.

**20** Daniel Cizma and Nati Linial. Irreducible nonmetrizable path systems in graphs. *Journal of Graph Theory*, 102(1):5–14, June 2022. doi:10.1002/jgt.22854.

**21** Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing for digraphs. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 885–886. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.315068.

**22** Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete applied mathematics*, 85(2):113–138, 1998.

**23** Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 333–341. ACM, 2019. doi:10.1145/3323165.3323177.

**24** Shang-En Huang and Seth Pettie. Thorup-zwick emulators are universally optimal hopsets. *Inf. Process. Lett.*, 142:9–13, 2019. doi:10.1016/j.ipl.2018.10.001.

**25** Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. *SIAM J. Discret. Math.*, 35(3):2129–2144, 2021. doi:10.1137/19M1306154.

**26** Ken-ichi Kawarabayashi and Yusuke Kobayashi. Breaking $o(n^{1/2})$-approximation algorithms for the edge-disjoint paths problem with congestion two. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 81–88. ACM, 2011. doi:10.1145/1993636.1993648.

**27** Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Stephan Kreutzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 70–78, 2014.

**28** Stavros G. Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using greedy algorithms and packing integer programs. In Robert E. Bixby, E. Andrew Boyd, and Roger Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization, 6th International IPCO Conference, Houston, Texas, USA, June 22-24, 1998, Proceedings*, volume 1412 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 1998. doi:10.1007/3-540-69346-7_12.

**29** Ilia Krasikov and Steven D. Noble. Finding next-to-shortest paths in a graph. *Inf. Process. Lett.*, 92(3):117–119, 2004. doi:10.1016/j.ipl.2004.06.020.

**30** William Lochet. A polynomial time algorithm for the $k$-disjoint shortest paths problem. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 169–178. SIAM, 2021. doi:10.1137/1.9781611976465.12.

**31** Raul Lopes and Ignasi Sau. A relaxation of the directed disjoint paths problem: A global congestion metric helps. *Theor. Comput. Sci.*, 898:75–91, 2022. doi:10.1016/j.tcs.2021.10.023.

**32** Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Comb.*, 7(4):365–374, 1987. doi:10.1007/BF02579324.

**33** Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms*, 4(3):29:1–29:17, 2008. doi:10.1145/1367064.1367069.

**34** Aviad Rubinstein and Virginia Vassilevska Williams. SETH vs approximation. *SIGACT News*, 50(4):57–76, 2019. doi:10.1145/3374857.3374870.

**35** Bang Ye Wu. A simpler and more efficient algorithm for the next-to-shortest path problem. In Weili Wu and Ovidiu Daescu, editors, *Combinatorial Optimization and Applications - 4th International Conference, COCOA 2010, Kailua-Kona, HI, USA, December 18-20, 2010, Proceedings, Part II*, volume 6509 of *Lecture Notes in Computer Science*, pages 219–227. Springer, 2010. doi:10.1007/978-3-642-17461-2_18.

# Axis-Parallel Right Angle Crossing Graphs

**Patrizio Angelini**
John Cabot University, Rome, Italy

**Michael A. Bekos**
Department of Mathematics, University of Ioannina, Greece

**Julia Katheder**
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany

**Michael Kaufmann**
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany

**Maximilian Pfister**
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Germany

**Torsten Ueckerdt**
Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

## Abstract

A RAC graph is one admitting a RAC drawing, that is, a polyline drawing in which each crossing occurs at a right angle. Originally motivated by psychological studies on readability of graph layouts, RAC graphs form one of the most prominent graph classes in beyond planarity.

In this work, we study a subclass of RAC graphs, called axis-parallel RAC (or apRAC, for short), that restricts the crossings to pairs of axis-parallel edge-segments. apRAC drawings combine the readability of planar drawings with the clarity of (non-planar) orthogonal drawings. We consider these graphs both with and without bends. Our contribution is as follows: (i) We study inclusion relationships between apRAC and traditional RAC graphs. (ii) We establish bounds on the edge density of apRAC graphs. (iii) We show that every graph with maximum degree 8 is 2-bend apRAC and give a linear time drawing algorithm. Some of our results on apRAC graphs also improve the state of the art for general RAC graphs. We conclude our work with a list of open questions and a discussion of a natural generalization of the apRAC model.

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

**Keywords and phrases** Graph drawing, RAC graphs, Graph drawing algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.9

**Related Version** *arXiv*: `https://arxiv.org/abs/2306.17073`

**Funding** The work of J. Katheder and M. Kaufmann is supported by DFG grant Ka 812-18/2.

## 1 Introduction

Planar graphs form a fundamental graph class in algorithms and graph theory. This is due to the fact that planar graphs have many useful properties, e.g., they are closed under minors and have a linear number of edges. Several decision problems, which are NP-complete for general graphs, become polynomial-time tractable, when restricted to planar inputs, e.g. [28]. As a result, the corresponding literature is tremendously large.

A recent attempt to extend this wide knowledge from planar to non-planar graphs was made in the context of *beyond-planarity*, informally defined as a generalization of planarity encompassing several graph-families that are close-to-planar in some sense (e.g., by imposing structural restrictions on corresponding drawings). Notable examples are the classes of





(i) $k$-planar graphs [31], in which each edge cannot be crossed more than $k$ times, (ii) $k$-quasi-planar graphs [1], which disallow $k$ pairwise crossing edges, and (iii) $k$-gap planar graphs [9], in which each crossing is assigned to one of the two involved edges such that each edge is assigned at most $k$ of its crossings. For an overview refer to the recent textbook [29].

While all of the aforementioned graph-classes are topological, meaning that the actual geometry of the graph's elements is not important, there is a single class proposed in the literature that is purely geometric. The motivation for its study primarily stems from cognitive experiments indicating that the negative effect of edge crossings in a graph drawing tends to be eliminated when the angles formed at the edge crossings are large [30]. In that aspect, the class of *right-angle-crossing* (RAC) graphs forms the optimal case in this scenario, where all crossing angles occur at 90°. Formally, it was introduced by Didimo, Eades and Liotta [22] a decade ago, and since then it has been a fruitful subject of intense research [5, 17, 19, 23, 25].

Generally speaking, the research on RAC graphs has focused on two main research directions depending on whether bends are allowed along the edges or not. Formally, in a $k$-bend RAC drawing of a graph each edge is a polyline with at most $k$ bends and the angle between any two crossing edge-segments is 90°. Accordingly, a $k$-bend RAC graph is one admitting such a drawing. A 0-bend RAC graph (or simply RAC graph) with $n$ vertices has at most $4n - 10$ edges [22], that is, at most $n - 4$ edges more than those of a corresponding maximal planar graph. The edge-density bounds for 1- and 2-bend RAC graphs are $5.5n - 10$ [2] and $74.2n$ [8], respectively, while for $k \geq 3$ it is known that every graph is $k$-bend RAC [25]. The research on RAC graphs, however, is not limited to edge-density bounds. Several algorithmic and combinatorial results [5, 7, 6, 18, 21, 25], as well as relationships with other graph classes [10, 13, 15, 16, 23, 14] are known; see [20] for a survey.

In this work, we continue the study of RAC graphs along a new and intriguing research line. Inspired by several well-established models for representing graphs (including the widely-used orthogonal model [12, 26, 27]), we introduce and study a natural subfamily of $k$-bend RAC graphs, which restricts all edge segments involved in crossings to be axis-parallel. We call this class $k$-bend apRAC. We expect that this restriction will further enhance the readability of the obtained drawings, as these combine the simple nature of the planar drawings with the clarity of the (non-planar) orthogonal drawings by allowing non axis-parallel edge segments, only when those are crossing-free. We further expect that our restriction will lead to new results of algorithmic nature. As a matter of fact, almost all algorithms that have been already proposed in the literature about $k$-bend RAC graphs in fact yield $k$-bend apRAC drawings [11, 22, 25]; e.g., every Hamiltonian degree-3 graph is 0-bend apRAC [6], while degree-4 and degree-6 graphs are 1- and 2-bend apRAC, respectively [3, 5].

Our contribution is as follows:

- In Section 2 we study preliminary properties of 0-bend apRAC graphs in order to prove that recognizing 0-bend apRAC graphs is NP-hard (see Theorem 3).
- We study whether $k$-bend apRAC graphs form a proper subclass of $k$-bend RAC graphs: For $k = 0$, we establish a strict inclusion relationship with $K_6$ minus one edge being the smallest graph separating the two classes. Further, our edge-density result for 1-bend apRAC graphs establishes a strict inclusion relationship for $k = 1$, see Corollary 5. The case $k = 2$ is more challenging (due to the degrees of freedom introduced by bends) and we leave it as an open problem. For $k \geq 3$, the two classes coincide, as the construction establishing that every graph is 3-bend RAC [22] can be converted to 3-bend apRAC by a rotation of 45°.

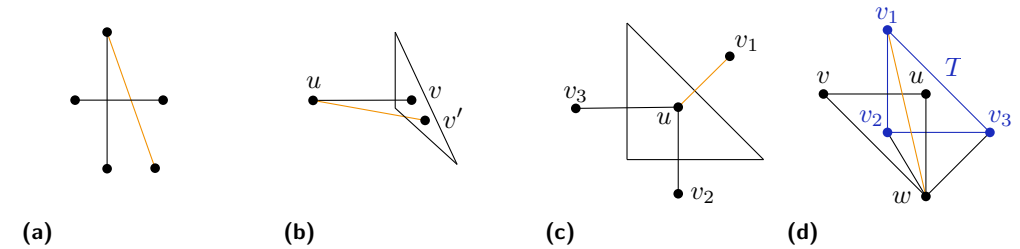



**Figure 1** Forbidden configurations by Properties 1–4.

- We establish bounds on the edge density of $n$-vertex $k$-bend apRAC graphs: For $k = 0$, we prove an upper bound of $4n - \sqrt{n} - 6$ and give a corresponding lower bound construction with $4n - 2\lfloor\sqrt{n}\rfloor - 7$ edges (see Theorem 1). For $k \in \{1, 2\}$, we give linear upper bounds that are tight up to small additive constants (see Theorems 4 and 6). Notably, for $k = 2$ our lower-bound construction is a graph with $n$ vertices and $10n - \mathcal{O}(1)$ edges. This bound extends to general 2-bend RAC graphs and improves the previous best one of $7.83n - \mathcal{O}(\sqrt{n})$ [8], answering an open question in [2].
- We show that every graph with maximum degree 8 is 2-bend apRAC and give a linear time drawing algorithm (see Theorem 8) improving the previous best known result stating that 7-edge colorable degree-7 graphs are 2-bend (ap)RAC [3].
- Inspired by the slope-number of graphs, in Section 7 we discuss a natural generalization of apRAC drawings where each edge segment involved in a crossing is parallel or perpendicular to a line having one out of $s$ different slopes.

## 2 Preliminaries

Throughout this paper, basic graph drawing concepts are used as found in [29, 32]. Let $G$ be a graph and $\Gamma$ be a polyline drawing of $G$ and let $e = (u, v)$ be an edge of $G$. We say that $e$ uses a horizontal (vertical) port at $u$ if the edge-segment of $e$ that is incident to $u$ is parallel to the $x$-axis (to the $y$-axis) in $\Gamma$. If $e$ uses neither a vertical nor a horizontal port at $u$, then it uses an *oblique* port at $u$. In particular, we denote the four orthogonal ports (i.e., the vertical and the horizontal ports) as $\{N, E, S, W\}$-ports according to compass directions. In a polyline drawing, vertices and bends are placed on grid-points, whereby the area of the drawing is determined by the smallest rectangular bounding box that contains the drawing. In the following, we recall two properties that hold for 0-bend RAC drawings.

▶ **Property 1** (Didimo, Eades and Liotta [22]). *In a 0-bend RAC drawing no edge is crossed by two adjacent edges (see Fig. 1a).*

▶ **Property 2** (Didimo, Eades and Liotta [22]). *A 0-bend RAC drawing does not contain a triangle $T$ formed by edges of the graph and two edges $(u, v)$ and $(u, v')$, such that $u$ lies outside $T$ while both $v$ and $v'$ lie inside $T$ (see Fig. 1b).*

Next, we establish two properties limited to 0-bend apRAC drawings.

▶ **Property 3.** *A 0-bend apRAC drawing does not contain a triangle $T$ formed by edges of the graph and three vertices $v_1, v_2, v_3$ adjacent to a vertex $u$, such that $v_1, v_2, v_3$ lie outside $T$ and $u$ lies inside $T$ (see Fig. 1c).*

**Proof.** Assuming the contrary, Property 1 implies that no two edges adjacent to $u$ cross the same boundary edge of $T$. Hence, $T$ consists of three axis-parallel edges; a contradiction. ◀

▶ **Property 4.** *Let $\Gamma$ be a 0-bend apRAC drawing containing a triangle $T$ formed by edges of the graph and two adjacent vertices $u$ and $v$ such that $u$ is contained inside $T$ while $v$ is outside $T$. Then, $\Gamma$ does not contain a vertex $w$ adjacent to $u$, $v$ and all vertices of $T$ (see Fig. 1d).*

**Proof.** For the sake of contradiction, assume there is a vertex $w$ adjacent to $u$, $v$ and all vertices of $T$. If $w$ is inside $T$ in $\Gamma$, then $(v, u)$ and $(v, w)$ violate Property 2; a contradiction. Otherwise, since $(u, v)$ and $(u, w)$ cross $T$, by Property 1, it follows that $T$ is a right-angled triangle whose legs are axis parallel. W.l.o.g., let $(v_1, v_2)$ and $(v_2, v_3)$ be the legs of $T$ crossed by $(u, v)$ and $(u, w)$, respectively, such that $(v_1, v_2)$ is horizontal and $(v_2, v_3)$ is vertical. It follows that the edge $(v_2, v_3)$ of $T$ is crossed by $(u, w)$ and $(w, v_1)$ violating Property 1; a contradiction. ◀

In Theorem 3 we leverage the following property shown in [7] of the so-called *augmented square antiprism graph*. The gadget used in the NP-hardness proof of Theorem 3 is depicted in Fig. 2a, while the vertex-colored subgraph in Fig. 2a corresponds to the augmented square antiprism graph.

▶ **Property 5** (Argyriou, Bekos, Symvonis [7])**.** *Any straight-line RAC drawing of the augmented square antiprism graph has two combinatorial embeddings.*

## 3 0-bend apRAC graphs

In this section, we focus on properties of 0-bend apRAC graphs. We start with an almost tight bound on the edge-density of 0-bend apRAC graphs - for comparison, recall that $n$-vertex 0-bend RAC graphs have at most $4n - 10$ edges [22].

▶ **Theorem 1.** *A 0-bend apRAC graph with $n$ vertices has at most $4n - \sqrt{n} - 6$ edges. Also, there is an infinite family of graphs with $4n - 2\lfloor\sqrt{n}\rfloor - 7$ edges that admit 0-bend apRAC drawings.*

**Proof.** For the upper bound consider any 0-bend apRAC drawing $\Gamma$ of a graph $G$ with $n$ vertices. As a $(k \times k)$-grid has only $k^2$ grid points, we may assume without loss of generality that the vertices of $G$ use at least $\sqrt{n}$ different $y$-coordinates in $\Gamma$. It follows that the subgraph $G_h$ of $G$ defined by the set $E_h$ of all horizontal edges of $\Gamma$ is a forest of paths with at least $\sqrt{n}$ components; at least one for each used $y$-coordinate. Thus $|E_h| \leq n - \sqrt{n}$. As $G - E_h$ is crossing-free in $\Gamma$, it has at most $3n - 6$ edges, giving the desired upper bound of $4n - \sqrt{n} - 6$ edges for $G$.

For the lower bound, consider the construction shown in Fig. 2b. For any even $k > 0$, construct a $k \times k$ grid graph $H_k$ which contains a pair of crossing edges in every quadrangular face. Let $G_k$ be the graph obtained from $H_k$ by adding two extremal adjacent vertices $N$ and $S$ connected to $2k - 1$ consecutive boundary vertices of $H_k$ each (refer to the blue edges in Fig. 2b and observe that the edge between $N$ and $S$ can be added by moving $N$ upwards and to the right and $S$ downwards and to the right of $H_k$). If we denote by $n$ the number of vertices of $G_k$, then $n = k^2 + 2$, $k = \sqrt{n-2}$ and thus $m = 4n - 2\lfloor\sqrt{n}\rfloor - 7$. ◀

Since there exist $n$-vertex 0-bend RAC graphs with $4n - 10$ edges, Corollary 2 follows from Theorem 1. In [4], we show that $K_6$ minus one edge is the smallest graph that is 0-bend RAC but not 0-bend apRAC.

▶ **Corollary 2.** *The class of 0-bend apRAC graphs is properly contained in the class of 0-bend RAC graphs.*

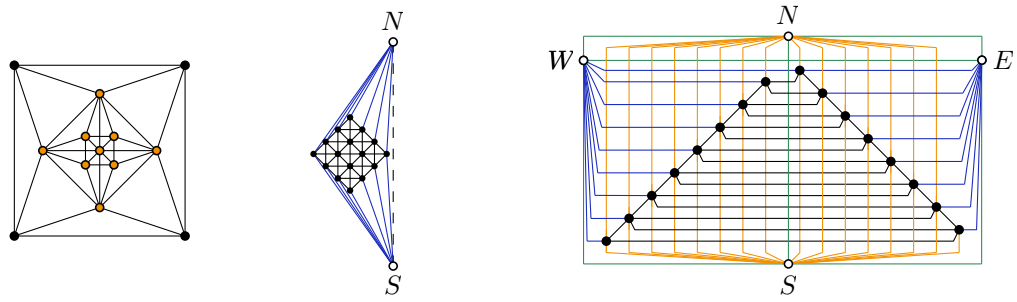



**Figure 2** (a) Graph used in Theorem 3. (b) Lower bound construction for 0-bend apRAC. (c) Lower bound construction for 1-bend apRAC.

We conclude this section by studying the recognition problem of whether a graph is 0-bend apRAC. Due to space reasons, we only sketch the idea of the proof; the complete proof can be found in [4].

▶ **Theorem 3.** *It is NP-hard to decide whether a given graph is* 0*-bend apRAC.*

**Sketch.** We adjust the NP-hardness reduction (from 3-SAT) for the general case of RAC graphs introduced in [7], whose main gadgets use a building block $\mathcal{H}$. We determine a 0-bend apRAC graph $H$ as a substitute for this building block which has the same properties as $\mathcal{H}$: $(i)$ $H$ has a unique embedding, $(ii)$ there are four vertices properly contained in its interior, which can be connected to vertices in its exterior by crossing a single boundary edge, $(iii)$ no edge can (completely) pass through $H$ without forming a fan crossing, and $(iv)$ $H$ can be extended horizontally or vertically maintaining properties $(i) - (iii)$. The graph shown in Fig. 2a satisfies all these criteria and can therefore be used for the reduction. ◀

## 4 1-bend apRAC graphs

In this section, we will establish an upper bound and an almost matching lower bound for the class of 1-bend apRAC graphs. Recall that $n$-vertex 1-bend RAC graphs have at most $5.5n - 10$ edges [2].

▶ **Theorem 4.** *A* 1*-bend apRAC graph with $n$ vertices has at most $5n - 8$ edges. Also, there is an infinite family of graphs with $5n - 16$ edges that admit* 1*-bend apRAC drawings.*

**Proof.** For the upper bound, consider a 1-bend apRAC drawing $\Gamma$ of an $n$-vertex graph $G$. Each edge segment in $\Gamma$ is either horizontal (h), vertical (v) or oblique (o). For $x, y \in \{h, v, o\}$, let $E_{xy}$ be the edges of $G$ with two edge segments of type $x$ and $y$. Then, $E_{hv}$, $E_{ho}$, $E_{vo}$ and $E_{oo}$ form a partition of the edge-set of $G$, assuming that edges that consist of only one $h$-, $v$- or $o$-segment are counted towards $E_{ho}$, $E_{vo}$ and $E_{oo}$, respectively. By construction, any crossing involves exactly one vertical and one horizontal segment. Hence, the subgraph of $G$ induced by $E_{ho} \cup E_{oo}$ is planar and contains at most $3n - 6$ edges. Further, as every segment is incident to a vertex and since any vertex is incident to at most two vertical segments, we have $|E_{vo} \cup E_{hv}| \leq 2n$. We can assume that the topmost vertex $v_t$ is incident to at most one vertical edge-segment, since the edge segment incident to $v_t$ that points upwards cannot be involved in a crossing with a horizontal edge-segment. Otherwise, the endpoint incident to this edge segment would contradict the fact that $v_t$ is topmost in $\Gamma$. Hence, it can be replaced by a steep oblique edge-segment without introducing new crossings. Analogous observations can be made for the bottommost vertex in $\Gamma$, which implies that $|E_{vo} \cup E_{hv}| \leq 2n - 2$. Thus, $|E| = |E_{ho}| + |E_{vo}| + |E_{hv}| + |E_{oo}| \leq 5n - 8$.

Our lower bound construction is as follows; see Fig. 2c. For $n \geq 7$, we arrange $n-4$ vertices forming a cycle along the two legs of an isosceles triangle with a horizontal base (outer black edges), such that the left leg has $\lfloor \frac{n-4}{2} \rfloor$ vertices while the right one has $\lceil \frac{n-4}{2} \rceil$. These $n-4$ vertices are further joined by a $y$-monotone path of $n-7$ edges (inner black edges). Two extremal vertices $N$ and $S$ above and below the triangle are connected to all $n-4$ vertices (orange edges). Similarly, two extremal vertices $W$ and $E$ to the left and right of the triangle are connected to all vertices of the left and right legs of the triangle respectively (blue edges); the topmost vertex of the right leg is also connected to $W$. Finally, we add six edges between the extremal vertices, which gives $n-4+n-6+3(n-4)+6 = 5n-16$ edges. ◀

Since there exists 1-bend RAC graphs with $5.5n-72$ edges [2], the following corollary is immediate.

▶ **Corollary 5.** *The class of* 1*-bend apRAC graphs is a proper subclass of the one of* 1*-bend RAC graphs.*

## 5 2-bend apRAC graphs

In Theorem 6, we provide an upper-bound for the edge density of 2-bend apRAC graphs together with a lower-bound construction which is tight up to an additive constant. Our result provides a stark contrast to the one for 2-bend RAC graphs, where the current best upper-bound on the number of edges of $n$-vertex graphs is $74.2n$ [8], while the previous best lower bound-construction contained only $7.83n - \mathcal{O}(\sqrt{n})$ [8] edges.

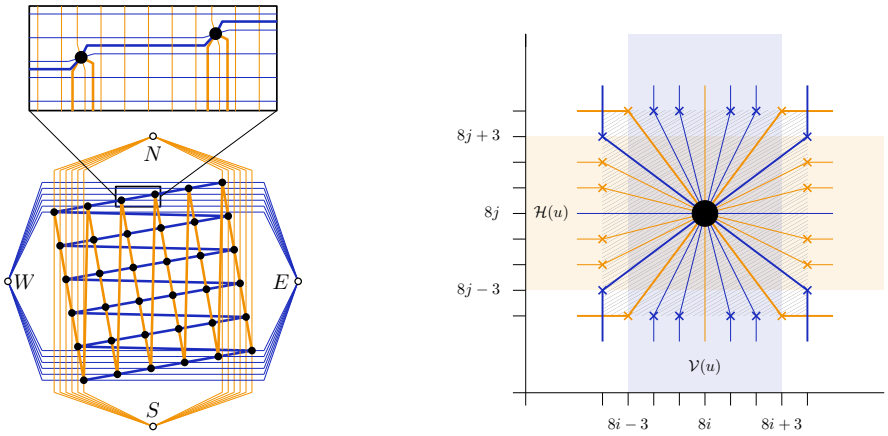
▶ **Theorem 6.** *A* 2*-bend apRAC graph with $n$ vertices has at most $10n-12$ edges. Also, there is an infinite family of graphs with $10n-46$ edges that admit* 2*-bend apRAC drawings.*

**Proof.** Consider a 2-bend apRAC drawing $\Gamma$ of an $n$-vertex graph $G$. Each edge segment in $\Gamma$ is either horizontal (h), vertical (v) or oblique (o). Denote by $S$ the set of edges that contain at least one segment in $\{h, v\}$ incident to a vertex. Since any vertex is incident to at most two vertical and at most two horizontal segments, it follows that $|S| \leq 4n$. Let $E_h$, $E_v$ and $E_o$ be the set of edges of $E \setminus S$ whose middle part is $h$, $v$ and $o$, respectively. Assuming that an edge of $E \setminus S$ consisting of less than three segments belongs to $E_o$, it follows that $E_h$, $E_v$ and $E_o$ form a partition of $E \setminus S$. Observe that the edges of $E_o$ cannot be involved in any crossing in $\Gamma$, as all of its segments are oblique. Further, no two edges of $E_h$ or of $E_v$ can cross. Hence, the subgraphs induced by $E_h \cup E_o$ and $E_v \cup E_o$ are planar and contain at most $3n-6$ edges each. Recall that $|S| \leq 4n$ and thus $|E| \leq |S| + |E_h| + |E_v| + 2|E_o| \leq 4n + 3n - 6 + 3n - 6 = 10n - 12$.

Refer to Fig. 3a for a schematization of the upcoming lower-bound construction and to Fig. 4 for a concrete example. Fix an integer $k \geq 6$ and consider a set $P$ of $k^2$ points of a $k \times k$ square grid in the plane but rotated very slightly, say counterclockwise, so that the points in each column have consecutive $x$-coordinates (consequently the points in each row have consecutive $y$-coordinates). For two points $p, q \in P$ let their *$x$-distance* $\mathrm{dist}_x(p, q)$ be the number of points in $P$ having their $x$-coordinate between $p$ and $q$. Similarly define the $y$-distance $\mathrm{dist}_y(p, q)$. The crucial property of point set $P$ is the following.

$$\text{For any } p \neq q \in P \text{ we have } \mathrm{dist}_x(p, q) + \mathrm{dist}_y(p, q) \geq k - 1 \geq 5. \tag{1}$$

Between any pair $p, q \in P$ with consecutive $x$-coordinates, i.e., $\mathrm{dist}_x(p, q) = 0$, we add a 2-bend edge with vertical middle segment by starting and ending with a very short oblique segment at $p$ respectively $q$. Similarly, we add a 2-bend edge with horizontal middle segment when $\mathrm{dist}_y(p, q) = 0$. Note that these are in total $2k^2 - 2$ edges, no two of which connect the same pair of points, due to (1).




(a) (b)

**Figure 3** (a) Illustration of the construction in Theorem 6 with $k = 6$. Edges with two oblique segments are indicated in orange for vertical middle segments and in blue for horizontal middle segments. Edges using the horizontal or vertical ports are omitted for readability. (b) Edge routing in the $8 \times 8$ box $B(u)$ of a vertex $u$. Blue ports are exclusively used by edges of $F_1$ and $F_3$ and orange ports by $F_2$ and $F_4$. Note that the ports illustrated by bold lines are reserved for oblique-2 edges. Bends on the border of the box are emphasized by a cross.

Next we add four additional points $N, E, S, W$ to the top, right, bottom, and left of all points in $P$, respectively. For every point $p$ we add a 2-bend edge with vertical middle segment between $p$ and $N$ starting with a very short oblique segment at $p$ and ending with an almost horizontal (but still oblique) segment at $N$. Similarly, we add a 2-bend edge with vertical middle segment between $p$ and $S$, as well as one with horizontal middle segment to each of $E, W$. Note that these are in total $4k^2$ edges, and that all oblique segments can be chosen such that all crossings involve middle segments only.

Next we add for (almost) each point $p \in P$ four more 2-bend edges. First, consider for $p$ the point $q \in P$ to the right of $p$ with $\text{dist}_x(p, q) = 1$, unless $p$ is one of the two rightmost points in $P$. We draw a 2-bend edge from $p$ to $q$ by starting with a horizontal segment at $p$ to almost the $x$-coordinate of $q$, continuing with a vertical segment to almost the $y$-coordinate of $q$, and ending with a very short oblique segment at $q$. Similarly, we use the left horizontal port at $p$ for an edge to the point $q$ left of $p$ with $\text{dist}_x(p, q) = 2$. (We take $x$-distance 2 instead of 1 to avoid introducing a parallel edge.) Symmetrically, we draw two edges using the vertical ports at $p$. Note that these are in total $4k^2 - 10$ edges, and that all crossings involve horizontal and vertical segments only.

Finally, we add easily add six edges to create a $K_4$ on vertices $N, E, S, W$. To conclude, we have constructed a 2-bend apRAC graph with $n = k^2 + 4$ vertices and $(2k^2 - 2) + 4k^2 + (4k^2 - 10) + 6 = 10k^2 - 6 = 10n - 46$ edges. ◀

## 6 Every graph with maximum degree 8 is 2-bend apRAC

In the following, we prove that graphs with maximum degree 8 admit 2-bend apRAC drawings of quadratic area which can be computed in linear time. We leverage the following result in order to decompose the input graph.

▶ **Lemma 7** (Eades, Symvonis, Whitesides [24]). *Let $G = (V, E)$ be an $n$-vertex undirected graph of degree $\Delta$ and let $d = \lceil \Delta/2 \rceil$. Then, there exists a directed multigraph $G' = (V, E')$ with the following properties:*

1. *each vertex of $G'$ has indegree $d$ and outdegree $d$;*
2. *$G$ is a subgraph of the underlying undirected graph of $G'$; and*
3. *the edges of $G'$ can be partitioned into $d$ edge-disjoint directed 2-factors (where a 2-factor is a spanning subgraph of $G'$ consisting of vertex disjoint cycles, called cycle cover in [24]).*

*The directed graph $G'$ and the $d$ 2-factors can be computed in $\mathcal{O}(\Delta^2 n)$ time.*

Now, we are ready to state the main result.

▶ **Theorem 8.** *Given a graph $G$ with maximum degree 8 and $n$ vertices, it is possible to compute in $\mathcal{O}(n)$ time a 2-bend apRAC drawing of $G$ with $\mathcal{O}(n^2)$ area.*

**Proof.** Let $G$ be a simple graph with maximum degree 8 and $n$ vertices. We apply Lemma 7 to augment $G$ to a **directed** 8-regular multigraph having four edge-disjoint 2-factors $F_1$, $F_2$, $F_3$ and $F_4$. Before we present our algorithm in full detail, we sketch an outline of the necessary steps. We want to stress that in the following, the direction of an edge $(u, v)$ plays an important role and hence we consider it as a directed edge with source $u$ and target $v$.

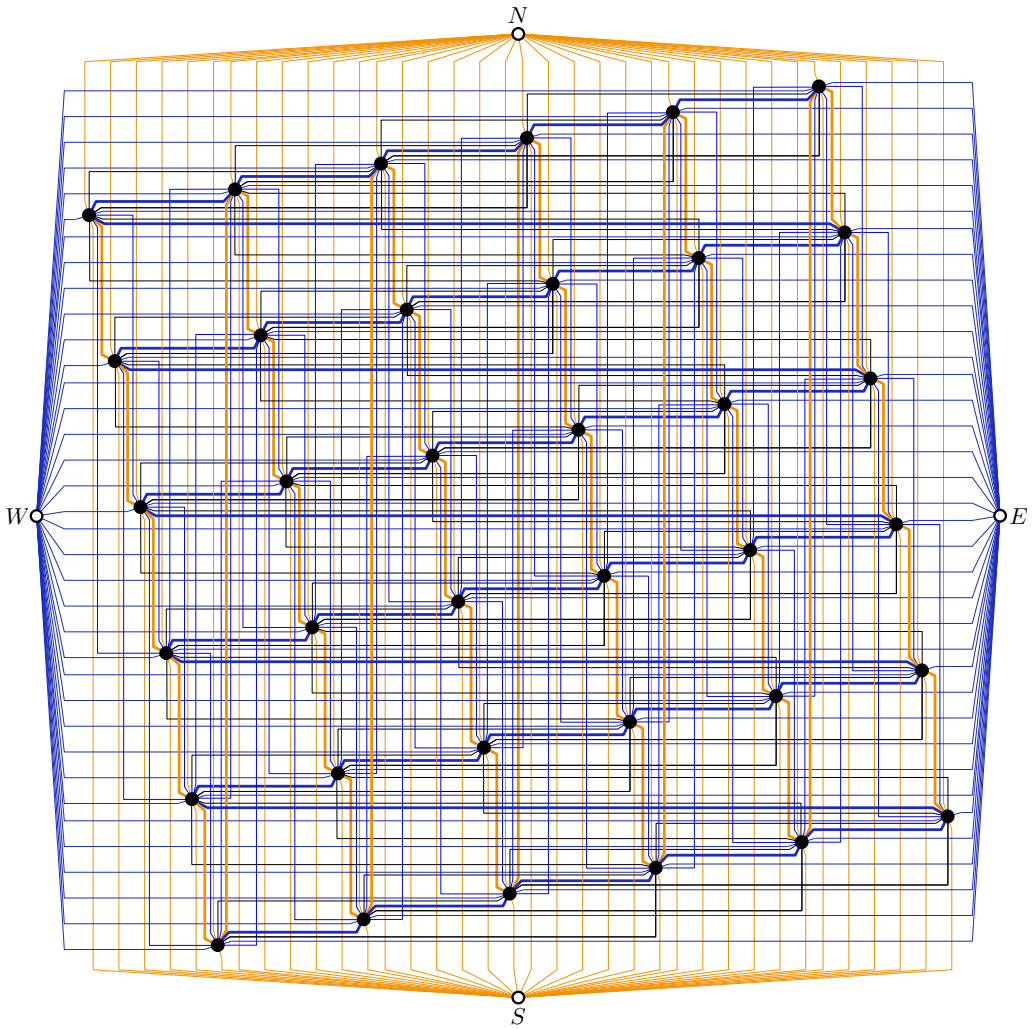



**Figure 4** Illustration of the construction in Theorem 6 with $k = 6$. The $K_4$ on the vertices $N, E, S, W$ is omitted due to space reasons.

## 6.1 Outline of the algorithm

In the first step, we will construct two total orders $\prec_x$ and $\prec_y$ of the vertices of $G$ which will determine the $x$- and $y$-coordinates of the vertices in the final drawing. In particular, if vertex $u$ of $G$ has the $i$-th position in $\prec_x$ and the $j$-th position in $\prec_y$, then $u$ will be placed at point $(8i, 8j)$ in the final drawing. We will construct these two orders independently such that $\prec_x$ is defined by $F_1 \cup F_3$ and $\prec_y$ is defined by $F_2 \cup F_4$. After the computation of $\prec_x$ and $\prec_y$, which finalizes the position of the vertices in our resulting drawing $\Gamma$, it remains to draw the edges which are fully characterized by the placement of the respective bend-points. Every edge will be drawn with exactly three segments, which are either horizontal, vertical or oblique. To ensure that all crossings in $\Gamma$ occur between horizontal and vertical segments, we will restrict oblique segments to be "short" (a precise definition follows below) and require that they are incident to a vertex. To this end, we will define, for each vertex $u$ of $G$, a closed box $B(u)$ centered at $u$ of size $8 \times 8$, such that the oblique segments incident to $u$ are fully contained inside $B(u)$. Note that by construction, the interior of two boxes do not overlap (they may touch at a corner). Since the $x$-coordinate of two consecutive vertices $u$ and $v$ of $\prec_x$ differs by exactly 8, there is a vertical line that is (partially) contained inside both $B(u)$ and $B(v)$ (analogous for a horizontal line and consecutive vertices in $\prec_y$). This allows us to join $u$ and $v$ by an edge that consists of two oblique segments, which is called an *oblique*-2 edge. If the unique orthogonal segment of an oblique-2 edge is vertical (horizontal), we will refer to it as a vertical (horizontal) oblique-2 edge. An edge that contains exactly one oblique segment will analogously be called an *oblique*-1 edge.

In the second step, we will classify every edge of $G$ as either an oblique-1 or an oblique-2 edge - again this classification is done independently for $F_1 \cup F_3$ and $F_2 \cup F_4$; we focus on the description of $F_1 \cup F_3$, the other one is symmetric. Let $e = (u, v)$ be an edge of $F_1 \cup F_3$. If $u$ and $v$ are consecutive in $\prec_x$, then $e$ is classified as a vertical oblique-2 edge. Otherwise, $e$ is classified as an oblique-1 edge such that the (unique) oblique segment is incident to the target $v$, while the orthogonal segment at $u$ uses the $E$-port at $u$ if $u \prec_x v$, otherwise it uses the $W$-port.

In the final step, we will specify the exact coordinates of the bend-points. At a high level, oblique segments (which are by construction all incident to vertices) will end at the boundary of the corresponding box, see Fig. 3b. The bend-points between vertical and horizontal segments are then naturally defined by the intersections of their corresponding lines.

The final drawing $\Gamma$ will then satisfy the following two properties.

- **(i)** No bend-point of an edge lies on another edge and
- **(ii)** the edges are drawn with two bends each so that only the edge segments that are incident to $u$ are contained in the interior of $B(u)$, while all the other edge segments are either vertical or horizontal.

This will guarantee that the resulting drawing is 2-bend RAC; for an example see Fig. 5. Note that (i) guarantees that no two segments have a non-degenerate overlap.

## 6.2 Computing $\prec_x$ and $\prec_y$

We will now describe how to construct $\prec_x$ and $\prec_y$ explicitly. We focus on the construction of $\prec_x$ which is based on $F_1$ and $F_3$, the order $\prec_y$ can be constructed analogously. Let $C_1, C_2, \ldots, C_k$ be an arbitrary ordering of the components of $F_1$. Recall that by definition, each such $C_i$ is a directed cycle. Let $S$ be a set of vertices that contains exactly one arbitrary vertex from each cycle in $F_1$ and let $P_1, P_2, \ldots, P_k$ be the resulting directed paths obtained by restricting the cycles to $V \setminus S$. Note that this may yield paths that are empty, i.e., when the

corresponding cycle consists of a single vertex. We construct $\prec_x$ (limited to $V \setminus S$) such that the vertices of each path appear consecutively defined by the unique directed walk from one endpoint of the path to the other. The relative order between paths is $P_1 \prec_x P_2 \prec_x \cdots \prec_x P_k$. Hence it remains to insert the vertices of $S$ into $\prec_x$. Throughout the algorithm, we will maintain the following invariant which will ensure the correctness of our approach.

**I.1** Let $u \in S$ be a vertex of cycle $C_i$. If $|C_i| > 1$, then $u$ is placed next to at least one vertex of $P_i$. Otherwise, $u$ is placed directly after the last vertex of $C_{i-1}$ (or as first vertex if $i = 1$) in $\prec_x$.

If I.1 is maintained, we can guarantee the following observation.

▶ **Observation 1.** *Let $u \in C_i$ and $v \in C_j$ be two vertices of $G$ with $i \neq j$. Then, the relative order of $u$ and $v$ in $\prec_x$ is known.*

Assume that each vertex in $S$ that belongs to $C_1, \ldots, C_{i-1}$ has been inserted in $\prec_x$. Let $u \in S$ be the vertex that belongs to $C_i \setminus P_i$. If $|C_i| \leq 2$, then we place $u$ immediately after the last vertex of $C_{i-1}$ in $\prec_x$ if $i > 1$, otherwise $u$ is the first vertex of $\prec_x$ which maintains I.1. Hence, in the remainder we can assume that $C_i$ consists of at least three vertices. Let $a$, $b$ and $c$ be the vertices of $G$ such that $(u, a)$, $(b, u) \in F_1$ and $(u, c) \in F_3$. Even though $G$ is a multigraph, we have that $a \neq b$ since $C_i$ contains at least three vertices. Hence, by construction we have $a \prec_x b$ - in particular, $a$ is the first vertex of $P_i$ in $\prec_x$, while $b$ is the last one. Let $C_j$ (possibly $j = i$) be the cycle that contains $c$. Note that it is possible that $c \in S$, i.e., $c$ is not part of $\prec_x$ initially. However, as this can only happen if $i \neq j$, we know the relative position of $u$ and $c$ by Observation 1. We distinguish between the following cases based on the relative order of cycle $C_i$ (which contains $u$) and cycle $C_j$ (which contains $c$) in $\prec_x$.

1. $j < i$. We insert $u$ immediately before $a$ in $\prec_x$ such that it is the first vertex of $C_i$, see Fig. 6a. Clearly, this maintains I.1.
2. $i < j$. This case is symmetric to the previous one - we insert $u$ immediately after $b$ in $\prec_x$ such that it is the last vertex of $C_i$, see Fig. 6b, which again maintains I.1.
3. $i = j$. In this case, we have that $c$ also belongs to $C_i$ (in particular, $c$ belongs to $P_i$ and thus is already part of $\prec_x$). If $c = a$ or $c = b$, we simply omit the edge $(u, c)$ and proceed as in the first case, i.e., we place $u$ as the first vertex of $C_i$. Otherwise, we insert $u$ directly before or directly after $c$ in $\prec_x$ based on the edge $(c, d) \in F_3$. The relative order of $c$ and $d$ in $\prec_x$ is known by Observation 1 unless $d \in C_i$. If $d \in P_i$, the relative order between $c$ and $d$ is also known (as both are already present in $\prec_x$). If $d \notin P_i$, then $d = u$ and we can omit the edge $(u, c) \in F_3$ (because it is a copy of $(c, d) \in F_3$), in which case we can again proceed as in the first case. Hence, $d \neq u$ holds. If $c \prec_x d$, we insert $u$ directly before $c$ in $\prec_x$, see see Fig. 6c, otherwise we insert $u$ directly after $c$ in $\prec_x$. In both cases, we maintain I.1.

This concludes our construction of $\prec_x$.

## 6.3 Classification of the edges and port assignment

We focus on the classification of the edges of $F_1 \cup F_3$ and their port assignment, the classification of the edges of $F_2 \cup F_4$ is analogous. Our classification will maintain the following invariants.

**I.2** The endpoints of each vertical oblique-2 edge are consecutive in $\prec_x$.

**I.3** Each oblique-1 edge $(u, v) \in F_1 \cup F_3$ is assigned the $W$-port at its source vertex $u$, if $v \prec_x u$; otherwise, if $u \prec_x v$, it is assigned the $E$-port at $u$.

**I.4** Every horizontal port is assigned at most once.

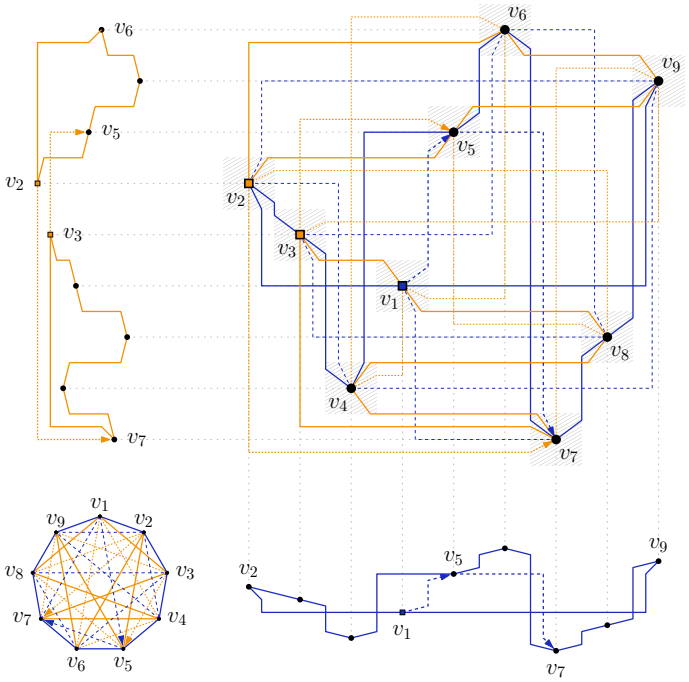



**Figure 5** A 2-bend apRAC drawing of $K_9$; $F_1$ and $F_3$ are blue; $F_2$ and $F_4$ are orange. Below the drawing of $K_9$ there is a illustration of the cycles in $F_1$ and the relevant edges in $F_3$ for positioning $v_1 \in S$ according to Case 3 in the construction of $\prec_x$. Similarly, a visualization of the cycles in $F_2$ and the relevant edges in $F_4$ is displayed to the left.

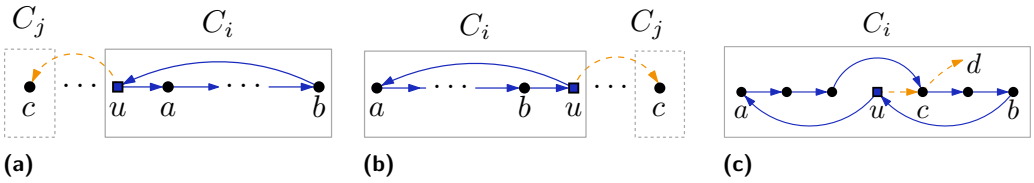



**Figure 6** Illustration of the construction of $\prec_x$, Case 1 is shown in (a), Case 2 in (b) and Case 3 in (c). Blue edges belong to $F_1$, while dashed orange edges belong to $F_3$.

Let us consider an edge $e \in F_1 \cup F_3$ between vertices $u$ and $v$. If $u$ and $v$ are consecutive in $\prec_x$, then we classify $e$ as a vertical oblique-2 edge. If $u$ and $v$ are not consecutive in $\prec_x$, we will classify $e$ as an oblique-1 edge, which therefore guarantees I.2. For any oblique-1 edge, we will, in an initial phase, assign the ports precisely as stated in I.3. In a subsequent step, we will create a unique assignment of the horizontal ports by reorienting some edges of $F_1 \cup F_3$ in order to guarantee I.4. Suppose that after the initial assignment, there exists a vertex $u$ such that one of its orthogonal ports is assigned to two oblique-1 edges. Assume first the $W$-port of $u$ is assigned to edges $(u, a)$ and $(u, b)$. By construction, $u$ has exactly one outgoing edge in $F_1$, say $(u, a)$, and exactly one outgoing edge in $F_3$, say $(u, b)$. Let $C_i$ be the cycle of $F_1$ that contains both $u$ and $a$ (which implies that $|C_i| > 1$, as we omit self-loops) and let $C_j$ be the cycle that contains $b$ (possibly $i = j$). Recall that by construction, the vertices of $P_i$ appear consecutively in $\prec_x$ before the insertion of the vertex $v \in C_i \setminus P_i$. Since $(u, a)$ is an oblique-1 edge, we have that $u$ and $a$ are not consecutive in $\prec_x$. If $|C_i| = 2$, one of $u$ or $a$ coincides with $v$, but then $u$ and $a$ are consecutive in $\prec_x$ and thus the edge $(u, a)$ is

an oblique-2 edge. Hence, $|C_i| > 2$ holds and we either have $u = v$, $a = v$ or $v$ was inserted directly in between $a$ and $u$. In the following, we will refer to Cases 1 - 3 of Section 6.2, where we computed the total order $\prec_x$.

1. $u = v$. Assume first that $C_i \neq C_j$. Then, since $(u, b)$ is assigned the $W$-port at $u$, we have $b \prec_x u$ by I.3 which implies $j < i$ and hence we placed $u$ according to Case 1, i.e., as the first vertex of $C_i$ in $\prec_x$. But since $a \in C_i$, we then have $u \prec_x a$ and thus $(u, a)$ would use the $E$-port at $u$, a contradiction.
   Hence assume that $C_i = C_j$, i.e., $b \in C_i$. Then we are in Case 3. In particular, we placed $u$ such that $u$ and $b$ are consecutive, thus $(u, b)$ is classified as an oblique-2 edge, again we obtain a contradiction.
2. $a = v$. Since $(u, a)$ uses the $W$-port at $u$ by assumption, we have that $a \prec_x u$ by I.3 and thus $a$ cannot be the last vertex of $C_i$ in $\prec_x$, and so we are in Case 1 or 3. In Case 1, $a$ is placed as the first vertex of $C_i$ since there exists a vertex $a'$ with $a' \prec_x a$ such that $(a, a') \in F_3$. Further, $a$ is placed next to vertex $v'$ (i.e., the first vertex of $P_i$ in $\prec_x$) with $(a, v') \in F_1$ by construction. Then, we can redirect the edge $(u, a) \in F_1$ such that we can assign $(a, u)$ the $E$-port at $a$ which solves the conflict at $u$ and does not introduce a conflict at $a$ which guarantees I.4. In Case 3, $a$ was placed consecutive to vertex $a' \in P_i$ with $(a, a') \in F_3$. As $(u, a)$ uses the $W$-port at $u$, $u$ is necessarily the last vertex of $C_i$ in $\prec_x$. Since the other neighbor of $a$ in $F_1$ different from $u$ is the first vertex of $C_i$ in $\prec_x$, i.e., it precedes $a$ in $\prec_x$, we can again reorient the edge $(u, a)$ and assign the edge $(a, u)$ to the free $E$-port of $a$, solving the conflict at $u$ which guarantees I.4.
3. $v$ was inserted directly in between $a$ and $u$. In this case, we have that both $a$ and $u$ belong to $P_i$. Since we assume that $(u, a)$ uses the $W$-port at $u$, it follows that $a \prec_x u$ holds. But then by construction, the edge of $F_1$ that joins $a$ und $u$ is directed from $a$ to $u$ and we obtain a contradiction.

The case where the $E$-port of $u$ is assigned to two edges can be solved in a similar way; refer to [4] for details.
Observe that if an edge $(u, v)$ was redirected, then both $u$ and $v$ belong to the same cycle $C_i$ of $F_1$ and since this operation has to be performed at most once per cycle, it follows that they can be considered independently. So far, we have computed $\prec_x$ and classified every edge of $F_1 \cup F_3$ guaranteeing Invariants 1-I.4. Symmetrically, we can compute $\prec_y$ and classify every edge of $F_2 \cup F_4$ guaranteeing corresponding versions of Invariants 1-I.4; see [4] for details.

## 6.4 Bend placement

We begin by describing how to place the bends of the edges on each side of the box $B(u)$ of an arbitrary vertex $u$ based on the type of the edge that is incident to $u$, refer to Fig. 3b. Let $(x_u, y_u)$ be the coordinates of $u$ in $\Gamma$ that are defined by $\prec_x$ and $\prec_y$. Recall that the box $B(u)$ has size $8 \times 8$. Let $e$ be an edge incident to $u$. We focus on the case in which $e \in F_1 \cup F_3$, the other case in which $e$ belongs to $F_2 \cup F_4$ is handled symmetrically by simply exchanging $x$ with $y$, "top/bottom" with "right/left" and "vertical" with "horizontal" from the following description. By definition, $e$ is either an oblique-1 edge or a vertical oblique-2 edge. Suppose first that $e$ is an oblique-1 edge. If $e = (u, v)$, i.e., $e$ is an outgoing edge of $u$ in $F_1 \cup F_3$, then by Invariant I.3 edge $e$ uses either the $W$- or $E$-port at $u$. In the former case, the segment of $e$ incident to $u$ passes through point $(y_u, y_u - 4)$, while in the latter case it passes through point $(y_u, y_u + 4)$. For an example, refer to the outgoing edge $(v_3, v_6)$ of $v_3$ in Fig. 5. If $e = (v, u)$, i.e., $e$ is an incoming edge of $u$ in $F_1 \cup F_3$, then by Invariant I.3 $e$ uses a horizontal port at $v$ and by the fact that every edge consists of exactly three segments, the

vertical segment of $e$ ends at the top or the bottom side of $B(u)$. Since any vertex has at most three incoming edges in $F_1 \cup F_3$ by construction, we can place the respective bends at $x$-coordinate $x_u + i$ with $i \in \{-2, -1, 1, 2\}$ and $y$-coordinate $y_u + 4$ ($y_u - 4$) for the top (bottom) side such that the assigned $i$-value is unique, refer to the incoming edge $(v_4, v_9)$ of $v_9$ in Fig. 5, where $i = -1$. Finally, the other bend-point of $e$ is uniquely defined as $(x_u + i, y_v)$, since it connects a vertical with a horizontal segment by construction.

Suppose now that $e$ is a vertical oblique-2 edge. By I.2, $u$ and $v$ are consecutive in $\prec_x$. If $v \prec_x u$ the $x$-coordinate of the bend point is $x_u - 4$, otherwise it is $x_u + 4$; e.g., refer to the edges $(v_2, v_3)$ and $(v_3, v_4)$ of $v_3$ in Fig. 5, respectively. In order to define the $y$-coordinate of the bend point, we have to consider the relative position of $u$ and $v$ in $\prec_y$. If $v \prec_y u$ the $y$-coordinate of the bend point of $e$ is $y_u - 3$ and otherwise it is $y_u + 3$. I.2 implies that any vertex has at most two vertical oblique-2 edges since no vertex has more than two direct neighbors in $\prec_x$. From the description of the bend-points, the observation follows:

▶ **Observation 2.** *Let $b$ be a bend-point that delimits an oblique segment $s$ which belongs to an edge $e$. If $s$ is incident to $u$, then $b$ does not lie on any other edge incident to $u$.*

## 6.5 Proof of correctness

The fact that the obtained drawing is 2-bend apRAC is proved in [4]. To complete the proof of Theorem 8, we discuss the time complexity and the required area. We apply Lemma 7 to $G$ to obtain $F_1$, $F_2$, $F_3$ and $F_4$ in $\mathcal{O}(n)$ time. For each cycle of $F_1$ and $F_2$, an appropriate ordering of its internal vertices, the classification of the incident edges and the assignment of the orthogonal ports can be computed in time linear in the size of the cycle. Clearly, computing the bend-points can be done in linear time as well. Hence we can conclude that the drawing can be computed in $\mathcal{O}(n)$ time. For the area, we can observe that the size of the grid defined by the boxes is $8n \times 8n$ and by construction, any vertex and any bend point is placed on a distinct point on the grid. ◀

# 7 Conclusion and Open Problems

In this paper, we introduced the class of $k$-bend apRAC graphs, gave edge-density bounds, studied inclusion relationships with the general $k$-bend RAC graphs, and concluded with an algorithmic result for graphs with maximum degree 8. A natural extension is to allow drawings where each crossing edge-segment is parallel or perpendicular to a line having one out of $s$ different slopes. We denote the class of graphs which admit such a drawing as $k$-bend $s$-apRAC, and w.l.o.g. we assume that the horizontal slope is among the $s$ ones. Observe that for $s = 1$, the derived class coincides with the class of $k$-bend apRAC graphs. By joining several copies of the graph supporting Property 5 that all share a common vertex, we show that 0-bend $s$-apRAC graphs form a proper subclass of 0-bend RAC graphs for any $s \in o(n)$; see [4] for details. We also adjust the proof of Theorem 6 to derive an upper bound on the edge density of 2-bend $s$-apRAC graphs, which is better than the one of [8] that holds for general 2-bend RAC graphs for values of $s$ up to 17. We conclude with the following open problems.

- Are there 2-bend RAC graphs that are not 2-bend apRAC?
- For $k \in \{1, 2\}$, our edge-density bounds do not relate to the simplicity of the drawings. Are bounds different for simple drawings, as in the general 1-bend RAC case [2]?
- For $k \in \{1, 2\}$, does the class of $k$-bend $s$-apRAC graphs on $n$ vertices coincide with the corresponding class of $k$-bend RAC graphs, when $s \in o(n)$?

- Based on our exploration of 2-bend apRAC graphs, we conjecture that the edge density of general 2-bend RAC graphs on $n$ vertices is $10n - \mathcal{O}(1)$.
- Another important open problem in the field is to settle the complexity of the recognition of general 1-bend RAC graphs. What if we restrict ourselves to the axis-parallel setting?
- From a practical perspective, can an advantage in the readability of $k$-bend apRAC drawings over $k$-bend RAC drawings be demonstrated in user studies?

## References


1 Pankaj K. Agarwal, Boris Aronov, János Pach, Richard Pollack, and Micha Sharir. Quasi-planar graphs have a linear number of edges. *Comb.*, 17(1):1–9, 1997. doi:10.1007/BF01196127.

2 Patrizio Angelini, Michael A. Bekos, Henry Förster, and Michael Kaufmann. On RAC drawings of graphs with one bend per edge. *Theor. Comput. Sci.*, 828-829:42–54, 2020. doi:10.1016/j.tcs.2020.04.018.

3 Patrizio Angelini, Michael A. Bekos, Julia Katheder, Michael Kaufmann, and Maximilian Pfister. RAC drawings of graphs with low degree. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *MFCS*, volume 241 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.MFCS.2022.11.

4 Patrizio Angelini, Michael A. Bekos, Julia Katheder, Michael Kaufmann, Maximilian Pfister, and Torsten Ueckerdt. Axis-parallel right angle crossing graphs. *CoRR*, abs/2306.17073, 2023. doi:10.48550/arXiv.2306.17073.

5 Patrizio Angelini, Luca Cittadini, Walter Didimo, Fabrizio Frati, Giuseppe Di Battista, Michael Kaufmann, and Antonios Symvonis. On the perspectives opened by right angle crossing drawings. *J. Graph Algorithms Appl.*, 15(1):53–78, 2011. doi:10.7155/jgaa.00217.

6 Evmorfia N. Argyriou, Michael A. Bekos, Michael Kaufmann, and Antonios Symvonis. Geometric RAC simultaneous drawings of graphs. *J. Graph Algorithms Appl.*, 17(1):11–34, 2013. doi:10.7155/jgaa.00282.

7 Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis. The straight-line RAC drawing problem is NP-hard. *J. Graph Algorithms Appl.*, 16(2):569–597, 2012. doi:10.7155/jgaa.00274.

8 Karin Arikushi, Radoslav Fulek, Balázs Keszegh, Filip Moric, and Csaba D. Tóth. Graphs that admit right angle crossing drawings. *Comput. Geom.*, 45(4):169–177, 2012. doi:10.1016/j.comgeo.2011.11.008.

9 Sang Won Bae, Jean-François Baffier, Jinhee Chun, Peter Eades, Kord Eickmeyer, Luca Grilli, Seok-Hee Hong, Matias Korman, Fabrizio Montecchiani, Ignaz Rutter, and Csaba D. Tóth. Gap-planar graphs. *Theor. Comput. Sci.*, 745:36–52, 2018. doi:10.1016/j.tcs.2018.05.029.

10 Michael A. Bekos, Walter Didimo, Giuseppe Liotta, Saeed Mehrabi, and Fabrizio Montecchiani. On RAC drawings of 1-planar graphs. *Theor. Comput. Sci.*, 689:48–57, 2017. doi:10.1016/j.tcs.2017.05.039.

11 Michael A. Bekos, Thomas C. van Dijk, Philipp Kindermann, and Alexander Wolff. Simultaneous drawing of planar graphs with right-angle crossings and few bends. *J. Graph Algorithms Appl.*, 20(1):133–158, 2016. doi:10.7155/jgaa.00388.

12 Therese C. Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom.*, 9(3):159–180, 1998. doi:10.1016/S0925-7721(97)00026-6.

13 Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani. Recognizing and drawing IC-planar graphs. *Theor. Comput. Sci.*, 636:1–16, 2016. doi:10.1016/j.tcs.2016.04.026.

14 Steven Chaplick, Henry Förster, Myroslav Kryven, and Alexander Wolff. Drawing graphs with circular arcs and right-angle crossings. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands*, volume 162 of *LIPIcs*, pages 21:1–21:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SWAT.2020.21.

15 Steven Chaplick, Fabian Lipp, Alexander Wolff, and Johannes Zink. Compact drawings of 1-planar graphs with right-angle crossings and few bends. *Comput. Geom.*, 84:50–68, 2019. doi:10.1016/j.comgeo.2019.07.006.

16 Hooman Reisi Dehkordi and Peter Eades. Every outer-1-plane graph has a right angle crossing drawing. *Int. J. Comput. Geom. Appl.*, 22(6):543–558, 2012. doi:10.1142/S021819591250015X.

17 Emilio Di Giacomo, Walter Didimo, Peter Eades, and Giuseppe Liotta. 2-layer right angle crossing drawings. *Algorithmica*, 68(4):954–997, 2014. doi:10.1007/s00453-012-9706-7.

18 Emilio Di Giacomo, Walter Didimo, Luca Grilli, Giuseppe Liotta, and Salvatore Agostino Romeo. Heuristics for the maximum 2-layer RAC subgraph problem. *Comput. J.*, 58(5):1085–1098, 2015. doi:10.1093/comjnl/bxu017.

19 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Henk Meijer. Area, curve complexity, and crossing resolution of non-planar graph drawings. *Theory Comput. Syst.*, 49(3):565–575, 2011. doi:10.1007/s00224-010-9275-6.

20 Walter Didimo. Right angle crossing drawings of graphs. In Seok-Hee Hong and Takeshi Tokuyama, editors, *Beyond Planar Graphs*, pages 149–169. Springer, 2020. doi:10.1007/978-981-15-6533-5_9.

21 Walter Didimo, Peter Eades, and Giuseppe Liotta. A characterization of complete bipartite RAC graphs. *Inf. Process. Lett.*, 110(16):687–691, 2010. doi:10.1016/j.ipl.2010.05.023.

22 Walter Didimo, Peter Eades, and Giuseppe Liotta. Drawing graphs with right angle crossings. *Theor. Comput. Sci.*, 412(39):5156–5166, 2011. doi:10.1016/j.tcs.2011.05.025.

23 Peter Eades and Giuseppe Liotta. Right angle crossing graphs and 1-planarity. *Discret. Appl. Math.*, 161(7-8):961–969, 2013. doi:10.1016/j.dam.2012.11.019.

24 Peter Eades, Antonios Symvonis, and Sue Whitesides. Three-dimensional orthogonal graph drawing algorithms. *Discret. Appl. Math.*, 103(1-3):55–87, 2000. doi:10.1016/S0166-218X(00)00172-4.

25 Henry Förster and Michael Kaufmann. On compact RAC drawings. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *ESA*, volume 173 of *LIPIcs*, pages 53:1–53:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.53.

26 Ulrich Fößmeier and Michael Kaufmann. Drawing high degree graphs with low bend numbers. In Franz-Josef Brandenburg, editor, *GD*, volume 1027 of *LNCS*, pages 254–266. Springer, 1995. doi:10.1007/BFb0021809.

27 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.

28 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM J. Comput.*, 4(3):221–225, 1975.

29 Seok-Hee Hong and Takeshi Tokuyama, editors. *Beyond Planar Graphs*. Springer, 2020. doi:10.1007/978-981-15-6533-5.

30 Weidong Huang, Peter Eades, and Seok-Hee Hong. Larger crossing angles make graphs easier to read. *J. Vis. Lang. Comput.*, 25(4):452–465, 2014. doi:10.1016/j.jvlc.2014.03.001.

31 János Pach and Géza Tóth. Graphs drawn with few crossings per edge. *Comb.*, 17(3):427–439, 1997. doi:10.1007/BF01215922.

32 Roberto Tamassia, editor. *Handbook on Graph Drawing and Visualization*. Chapman and Hall/CRC, 2013. URL: https://www.crcpress.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/9781584884125.

# (No) Quantum Space-Time Tradeoff for USTCON

**Simon Apers**
CNRS, IRIF, Paris, France

**Stacey Jeffery**
CWI & QuSoft, Amsterdam, The Netherlands

**Galina Pass**
Korteweg-de Vries Institute for Mathematics & QuSoft, University of Amsterdam, The Netherlands
Faculty of Computer Science, Ruhr University Bochum, Germany

**Michael Walter**
Faculty of Computer Science, Ruhr University Bochum, Germany

## Abstract

Undirected *st*-connectivity is important both for its applications in network problems, and for its theoretical connections with logspace complexity. Classically, a long line of work led to a time-space tradeoff of $T = \widetilde{O}(n^2/S)$ for any $S$ such that $S = \Omega(\log(n))$ and $S = O(n^2/m)$. Surprisingly, we show that quantumly there is no nontrivial time-space tradeoff: there is a quantum algorithm that achieves both optimal time $\widetilde{O}(n)$ and space $O(\log(n))$ simultaneously. This improves on previous results, which required either $O(\log(n))$ space and $\widetilde{O}(n^{1.5})$ time, or $\widetilde{O}(n)$ space and time. To complement this, we show that there is a nontrivial time-space tradeoff when given a lower bound on the spectral gap of a corresponding random walk.

**2012 ACM Subject Classification** Theory of computation → Quantum computation theory; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Undirected st-connectivity, quantum walks, time-space tradeoff

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.10

**Related Version** *Full Version*: https://arxiv.org/abs/2212.00094

**Funding** *Stacey Jeffery*: Supported by ERC STG grant 101040624-ASC-Q, NWO Klein project number OCENW.Klein.061, and ARO contract no W911NF2010327. SJ is a CIFAR Fellow in the Quantum Information Science Program.
*Galina Pass*: Supported by the National Agenda for Quantum Technologies (NAQT), as part of the Quantum Delta NL programme.
*Michael Walter*: Supported by the European Research Council (ERC) through ERC Starting Grant 101040907-SYMOPTIC, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, the Federal Ministry of Education and Research (BMBF) through project Quantum Methods and Benchmarks for Resource Allocation (QuBRA), and NWO grant OCENW.KLEIN.267.

**Acknowledgements** Part of this work was initiated at the 2022 QOPT (QuantERA ERA-NET Cofund 2022-25) workshop hosted at Université Libre de Bruxelles.

## 1 Introduction

For an undirected graph $G = (X, E)$ on $n = |X|$ vertices and $m = |E|$ edges, with $s, t \in X$, *st*-connectivity or USTCON is the problem of deciding whether $s$ and $t$ are in the same component. This problem has applications in many other graph and network problems, and is of theoretical importance for its connection with space complexity (see e.g. [23]). In particular, USTCON is complete for the class *symmetric logspace*, SL, which was shown to be equal to *logspace*, L, by exhibiting a classical deterministic logspace algorithm for USTCON [22]. In this paper, we consider quantum algorithms for this problem.


There are different versions of the problem USTCON depending on how $G$ is accessed. If $G$ is given as an adjacency matrix, we denote the problem $\text{USTCON}_{\text{mat}}$. If $G$ is given as an array of arrays, one for each vertex, enumerating the neighbours, we denote the problem $\text{USTCON}_{\text{arr}}$. [1] If one only cares about space complexity, these problems are equivalent, but the same is not true of time complexity: adjacency queries can simulate an array query, and vice versa, in logspace, but there is a non-negligible time overhead.

A classical deterministic algorithm based on breadth-first search or depth-first search can solve $\text{USTCON}_{\text{arr}}$ in $\widetilde{O}(m)$ time, using $\widetilde{O}(n)$ space. Using a random walk, the space complexity can be improved to $O(\log(n))$, at the expense of $\widetilde{O}(nm)$ time complexity [2]. A series of works [11, 7, 14, 5, 16] culminated in a space-time tradeoff for $\text{USTCON}_{\text{arr}}$ of $T = \widetilde{O}(n^2/S)$ queries for any space bound $S = \Omega(\log(n))$ and $S = O(n^2/m)$, due to Kosowski [19]. While there is no matching time-space lower bound, it is unlikely that this tradeoff can be significantly improved (see [19, Section 5.1 of arXiv v2] for a discussion). Kosowski's algorithm is based on using Metropolis-Hastings random walks to find connections between $S$ sampled vertices and $s, t$ until it is becomes possible to conclude that $s$ and $t$ are connected. For comparison, in the adjacency matrix model, the randomized query complexity of $\text{USTCON}_{\text{mat}}$ is $\widetilde{\Theta}(n^2)$ and there is no space-time tradeoff.

A quantum algorithm of Dürr, Heiligman, Høyer and Mhalla [13] for CONNECTIVITY can be adapted to solve $\text{USTCON}_{\text{mat}}$ in $\widetilde{O}(n^{1.5})$ time and $\text{USTCON}_{\text{arr}}$ in $\widetilde{O}(n)$ time, both of which are optimal up to polylog factors. Both of these algorithms use $\widetilde{O}(n)$ space, of which all but $O(\log(n))$ can be classical space (assuming quantum RAM access). A subsequent quantum algorithm for $\text{USTCON}_{\text{mat}}$ due to Belovs and Reichardt uses $\widetilde{O}(n^{1.5})$ time, but only $O(\log(n))$ space [9], which is optimal in terms of both space and time. It is also possible to solve $\text{USTCON}_{\text{arr}}$ in $O(\log(n))$ space and $\widetilde{O}(\sqrt{nm})$ time, using a quantum walk (see for example [8]). This quantum walk algorithm requires a quantum version of array access to the input graph, which we refer to as $\text{USTCON}_{\text{qw}}$ in the next section.

## 1.1 Summary of results

We describe new quantum walk algorithms for $\text{USTCON}_{\text{arr}}$. These algorithms consider a quantum walk version of the adjacency array model, in which the input graph is accessed by a quantum analogue of classical random walk steps. Recall that in the adjacency array model, we assume that for any vertex $u$, we can query, for any $i \in [d_u]$, the $i$-th neighbour of $u$, $v_i(u)$. Then a random walk step can be performed from state $u$ by sampling a uniform $i \in [d_u]$, and then computing $v_i(u)$, which becomes the current state. In the *quantum walk access model*, we assume that for any vertex $u$, we can prepare a uniform superposition over the neighbours of $u$. While these models are not identical, they are very similar, and in Section 3, we formally define the models, and show that quantum walk access can be simulated in the array model with polylogarithmic overhead under reasonable additional assumptions.

Letting $\text{USTCON}_{\text{qw}}$ denote the *st*-connectivity problem in the quantum walk access model, we present a one-sided error quantum algorithm that solves $\text{USTCON}_{\text{qw}}$ in time $\widetilde{O}(n)$ and space $O(\log(n))$. Perhaps surprisingly, this means that $\text{USTCON}_{\text{qw}}$ admits *no* nontrivial tradeoff between space and time in the quantum setting – a single algorithm can solve this problem optimally in terms of both time and space (see Theorem 15 for the formal result).

[1] There are variations on the details of this model. For now, we allow $\text{USTCON}_{\text{arr}}$ to stand in for multiple variations of the array access model, but precise details of the variations can be found in Section 3.

**Table 1** A summary of classical (randomized) and quantum time and space complexities for USTCON in the adjacency matrix and adjacency array models. The classical results for $\text{USTCON}_{\text{mat}}$ follow from (1) the $\log(n)$-space result for $\text{USTCON}_{\text{arr}}$ with an $n/d$ overhead for finding neighbours of the current vertex in a $d$-regular graph; and (2) BFS.

| $\text{USTCON}_{\text{mat}}$ | | |
|---|---|---|
| | Time | TS-tradeoffs |
| Classical | $\widetilde{\Theta}(n^2)$ | $S = O(\log(n))$, $T = \widetilde{O}(n^3/d)$<br>$S = \widetilde{O}(n)$, $T = \widetilde{O}(n^2)$ |
| Quantum | $\widetilde{\Theta}(n^{1.5})$ | $S = O(\log(n))$, $T = \widetilde{O}(n^{1.5})$ [9] |
| $\text{USTCON}_{\text{arr}}$ | | |
| | Time | TS-tradeoffs |
| Classical | $\widetilde{\Theta}(m)$ | $T = \widetilde{O}(\max\{n^2/S, m\})$ |
| Quantum | $\widetilde{\Theta}(n)$ | $S = O(\log(n))$, $T = \widetilde{O}(n^{1.5})$<br>$S = T = \widetilde{O}(n)$ [13] |
| | | **This work:** $S = O(\log(n))$, $T = \widetilde{O}(n)$ |

▶ **Theorem 1** (Informal). *There is a $O(\log(n))$-space quantum algorithm that decides $\text{USTCON}_{\text{qw}}$ with one-sided error in $\widetilde{O}(n)$ time.*

In this paper, when we say *time*, we are counting: (1) quantum gates (unitaries that act on at most a constant number of qubits); (2) quantum walk queries to $G$; and (3) (quantum) random access (QCRAM) operations (QCRAM is used in our second algorithm only, see below). Inspired by [19], our algorithm is based on a quantum walk search for $t$ starting from $s$ using a random walk that can be interpreted as a Metropolis-Hastings random walk.

Because of the close relationship between USTCON and classical logspace, we can consider what this means for logspace problems in general. It does not mean that more space does not reduce the quantum time complexity of *any* problem, but it is interesting to consider: in what settings do we get a non-trivial time-space tradeoff? We consider one such setting: when we are given a promise on the spectral gap or mixing time of the random walk on $G$ (see Section 2.2). In that case, we prove the following theorem (see Theorem 17 for the formal result).

▶ **Theorem 2** (Informal). *Suppose whenever $s$ and $t$ are connected, the random walk spectral gap is at least $\delta > 0$. For any $S \in \Omega(\log(n))$, there is a quantum algorithm that decides $\text{USTCON}_{\text{qw}}$ with bounded error in $O(S)$ space and $T = \widetilde{O}\left(\frac{S}{\delta} + \sqrt{\frac{n}{\delta S}}\right)$ time.*

The time bound decreases monotonically for $S \in \Omega(\log n)$ until $S \in O((n\delta)^{1/3})$, at which point it reaches time complexity $T = \widetilde{O}(n^{1/3}/\delta^{2/3})$. We leave it as an open problem to prove a matching lower bound (at least for some values of $\delta$), which would prove that in certain regimes, it is not possible to achieve optimal time and space simultaneously.

Our algorithm takes inspiration from [3]. In fact, with some imagination, one can derive a similar (but incomparable) time-space tradeoff for $\text{USTCON}_{\text{qw}}$ from that work: for $1 \leq S \leq m$, the algorithm in [3] can be adapted to use space $\widetilde{O}(S)$ and time $T \in \widetilde{O}(S + \sqrt{m/(\delta S)})$, with $\delta$ the random walk spectral gap.

In the space bound $S$ of both algorithms in Theorems 1 and 2, only $O(\log(n))$ memory needs to be actual quantum workspace (i.e., qubits). The remaining $O(S)$ memory can be classical RAM in the first algorithm and QCRAM in the second algorithm, that is, classical RAM that is queryable at a quantum superposition of addresses. We discuss the latter in Section 2.4.

We summarize our results in Figure 1. For $S = \log(n)$, the algorithm of Theorem 2 has a worse time complexity than the algorithm of Theorem 1, whenever $\delta < \frac{1}{n}$. We leave it as an open problem to give a single algorithm that is optimal for all $\delta$.

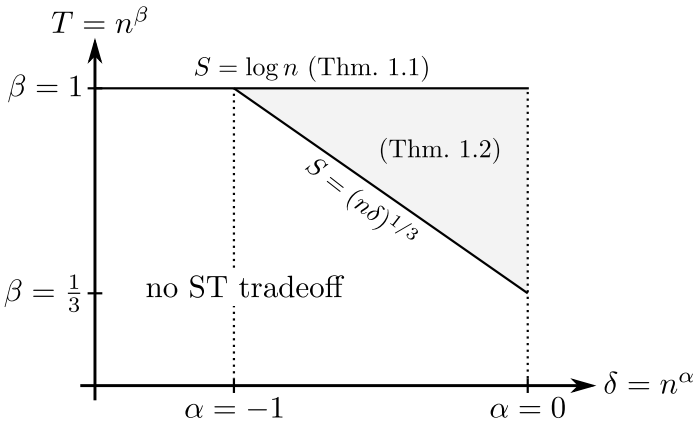



**Figure 1** Quantum space-time tradeoffs for USTCON, with axes representing the time complexity and spectral gap promise (up to polylog-factors). The grey area represents the regime in which a non-trivial tradeoff is achieved. Theorem 1 (upper line) corresponds to the regime with space $S = O(\log n)$ and time $T = \widetilde{O}(n)$. Theorem 2 (grey area) corresponds to the regime with a promise on $\delta$, and interpolates between $S = O(\log n)$ and $T = \widetilde{O}(n)$, and $S = O((n\delta)^{1/3})$ and $T = \widetilde{O}(n^{1/3}/\delta^{2/3})$.

### Organization

The remainder of this paper is organized as follows. We describe preliminaries in Section 2 and Section 3. In Section 4, we prove Theorem 1 by exhibiting a quantum algorithm for $\text{USTCON}_{\text{qw}}$ that is optimal in both time and space. For completeness, we also include a proof of a corresponding lower bound in Section 4.1. In Section 5, we prove Theorem 2 exhibiting a quantum time-space tradeoff when given a promise on the spectral gap.

## 2 Preliminaries

We first give some general notation. For a positive integer $k$, we let $[k] = \{1, \ldots, k\}$. Throughout this work, $n$ denotes the number of vertices and $m$ the number of edges of the input graph. For any function $f$, we let $\widetilde{O}(f(n)) = f(n) \cdot \text{polylog}(n)$.

### 2.1 Probability theory

A *(probability) distribution* on a finite set $X$ is a non-negative function $\sigma\colon X \to \mathbb{R}_{\geq 0}$ such that $\sum_{v \in X} \sigma(v) = 1$. Its *support* is defined as $\text{supp}(\sigma) := \{v \in X : \sigma(v) > 0\}$. We will implicitly identify such $\sigma$ with *row* vectors, as is customary in the random walk literature. To any distribution $\sigma$, we also associate a quantum state $|\sigma\rangle := \sum_{v \in X} \sqrt{\sigma(v)}\,|v\rangle$. Measuring $|\sigma\rangle$ in the standard basis returns a sample from $\sigma$.

For any distribution $\sigma$ on $X$, and any subset $M \subseteq X$, we will let $\sigma(M) = \sum_{u \in M} \sigma(u)$. We let $\sigma_M$ denote the *normalized restriction* of $\sigma$ to $M$, defined by $\sigma_M(u) = \sigma(u)/\sigma(M)$ for all $u \in M$ and $\sigma_M(u) = 0$ elsewhere.

Finally, the *total variation distance* between two distributions $\sigma$ and $\tau$ on $X$ is defined as

$$\|\sigma - \tau\|_{\mathrm{TV}} := \frac{1}{2} \sum_{u \in X} |\sigma(u) - \tau(u)| = \max_{A \subseteq X} |\sigma(A) - \tau(A)|.$$

## 2.2 Random walks

Fix an undirected graph $G = (X, E)$ with $n = |X|$ vertices and $m = |E|$ edges. We take $E \subseteq \binom{X}{2}$, that is, edges $e \in E$ are subsets $e = \{u, v\} = \{v, u\}$ of pairs of vertices. We will let

$$N(u) := \{v \in X : \{u, v\} \in E\}$$

denote the neighbourhood of $u \in X$, and $d_u = |N(u)|$ the degree of $u$. For convenience we assume that all vertices have positive degree.

Fix edge weights given by a symmetric matrix $W \in \mathbb{R}_{\geq 0}^{X \times X}$ such that $W_{u,v} = W_{v,u}$ for all $u, v \in X$, and $W_{u,v} > 0$ if and only if $\{u, v\} \in E$. Then $G = (X, E, W)$ defines a *weighted graph*. When no $W$ is given, the graph is *unweighted* and we let $W_{u,v} = 1$ for all $\{u, v\} \in E$. For $u \in X$, define $w_u = \sum_{v \in X} W_{u,v}$. The corresponding *(weighted) random walk* is the reversible Markov chain on $X$ with *transition matrix* $P \in \mathbb{R}_{\geq 0}^{X \times X}$ given by

$$P_{u,v} = \begin{cases} \frac{W_{u,v}}{w_u} & \text{if } \{u, v\} \in E \\ 0 & \text{otherwise} \end{cases} \qquad \forall u, v \in X. \tag{1}$$

This means that the probability of moving from the vertex $u$ along an edge to a neighbouring vertex $v$ is proportional to the edge's weight. In the unweighted case, this is called the *simple random walk*; in each step it simply moves to a neighbouring vertex chosen uniformly at random.

Let $\pi \in \mathbb{R}_{>0}^{X}$ be the distribution defined by

$$\pi(u) = \frac{w_u}{\mathcal{W}(G)} \qquad \forall u \in X,$$

where $\mathcal{W}(G) = \sum_{u \in X} w_u = \sum_{u,v \in X} W_{u,v}$. In the unweighted case, $\pi$ is proportional to the degree. The distribution $\pi$ is a *stationary* distribution of the random walk, i.e., $\pi P = \pi$ (it is a left eigenvector of $P$ with eigenvalue 1).

In fact, when the graph $G$ is connected, $\pi$ is also the *unique* stationary distribution of $P$. If in addition the graph is not bipartite, then all other eigenvalues have absolute value strictly less than one. That is, if $1 = \lambda_1 \geq \cdots \geq \lambda_n \geq -1$ are the eigenvalues of $P$ then the *(absolute) spectral gap* $\gamma_\star = \gamma_\star(G) := \min\{1 - |\lambda_j| : j = 2, \ldots, n\} = \min\{1 - \lambda_2, 1 + \lambda_n\}$ is strictly positive. Importantly, the inverse of the spectral gap bounds the random walk's *mixing time*, that is, the time required for convergence to the stationary distribution:

▶ **Theorem 3** ([21, Thm. 12.4]). *Assume $G$ is connected and not bipartite. Let $\varepsilon > 0$ and*

$$t \geq \frac{1}{\gamma_\star} \log\left(\frac{1}{\varepsilon \pi_{\min}}\right),$$

*where $\pi_{\min} = \min_{u \in X} \pi(x)$. Then $\|\sigma P^t - \pi\|_{\mathrm{TV}} \leq \varepsilon$ for any distribution $\sigma$ on $X$.*

Conversely, it is known that $t \geq (\frac{1}{\gamma_\star} - 1)\log\left(\frac{1}{2\varepsilon}\right)$ is necessary to ensure mixing from an arbitrary initial distribution [21, Thm. 12.5]. In the unweighted case, we have $\pi_{\min} \geq \frac{d_{\min}}{n\, d_{\max}} \geq \frac{1}{n^2}$, so the former shows that Theorem 3 is tight up to $\log(n)$ factors in that case.

Finally, for any $s, t \in X$ we let $\mathcal{H}_{s,t}$ denote the *hitting time* from $s$ to $t$, which is the expected number of steps needed to reach $t$ in a random walk starting from $s$. We let $\mathcal{C}_{s,t} = \mathcal{H}_{s,t} + \mathcal{H}_{t,s}$ denote the *commute time* between $s$ and $t$ – the expected number of steps needed to reach $t$ and then return to $s$ in a random walk starting from $s$. These quantities are finite if and only if $s$ and $t$ are in the same component of $G$. More generally, the commute time $\mathcal{C}_{s,M}$ from $s$ to a subset $M \subseteq X$ is the expected number of steps needed to reach any vertex in $M$ and then return to $s$ in a random walk starting from $s$.

## 2.3 Quantum walk search algorithms

Quantum walk search refers to the use of quantum walks to find certain "marked" elements on a graph. We will use quantum walk search to search for a vertex connected to $t$ in the connected component of $S$. Specifically, we will use the following special case of [4, Thm. 13].[2]

▶ **Theorem 4.** *Let $P$ be a random walk on a weighted graph with vertex set $X$, $M \subseteq X$ a subset of "marked" vertices, and $s \in X$. Let $\mathsf{C}$ be the (quantum) time complexity to check for a given $u \in X$ whether $u \in M$, let $\mathsf{U}$ be the time complexity of implementing the weighted quantum walk oracle*

$$|u\rangle|0\rangle \mapsto \sum_{v \in N(u)} \sqrt{P_{u,v}}\,|u\rangle|v\rangle\,.$$

*in space $O(\log(n))$. Let $\mathcal{C}$ be a known upper bound on the commute time $\mathcal{C}_{s,M}$ in the case where $s$ and $M$ are connected (and in particular $M \neq \emptyset$). Then there is a quantum algorithm that, if $M \neq \emptyset$ and $s$ is connected to $M$, finds an element of $M$ with probability at least $2/3$. If $M = \emptyset$ or $s$ is not connected to $M$, then the algorithm outputs a vertex not in $M$. The algorithm has time complexity $O(\sqrt{\log(\mathcal{C})}\log(n) + \sqrt{\mathcal{C}\log(\mathcal{C})\log(\log(\mathcal{C}))}(\mathsf{C}+\mathsf{U}))$ and space complexity $O(\log(n))$.*

## 2.4 Quantum RAM

Our algorithm will exploit the given space by saving sets of vertices which will be either connected to $s$ or to $t$. For our quantum algorithm to access this space, we assume access to a so-called quantum-classical random access memory or *QCRAM*. This refers to a memory that only stores classical information, but can be queried at a superposition of addresses. More specifically, an $R$-bit QCRAM stores a string of bits $q \in \{0,1\}^R$ so that the following operations are supported in time $\mathrm{polylog}(R)$:

1. *UPDATE(i, x):* store $x \in \{0,1\}$ in the $i$-th bit (i.e., set $q_i = x$).
2. *QUERY:* for any superposition $\sum_i \alpha_i |i\rangle|s_i\rangle$, it maps

$$\sum_i \alpha_i |i\rangle|s_i\rangle \mapsto \sum_i \alpha_i |i\rangle|s_i \oplus q_i\rangle\,.$$

As was first described by Kerenidis and Prakash [18], using such a QCRAM we can set up a data structure to generate quantum superpositions over elements in the QCRAM. We will use the following formulation based on [3].

[2] To see that this follows from [4, Thm. 13], note that when $|\sigma\rangle = |s\rangle$, the cost to set up $|\sigma\rangle$ is $\log(n)$ and the value $C_{\sigma,M}$ from [4] is exactly the commute time from $s$ to $M$ [4, Thm. 4].

▶ **Lemma 5.** *Fix integer parameters $\ell$ and $k$. Using an $O(k\ell\log(\ell))$-bit QCRAM, there is a data structure, $D$, that stores up to $\ell$ elements $x \in \{0,1\}^k$ with associated integer weights, $c_x$, of bounded absolute value for some poly($\ell$) bound, and supports the following operations in time $O(k \cdot \mathrm{polylog}(k\ell))$ per operation:*

1. *insertion or deletion of a pair $(x, c_x)$,*
2. *quantum queries of the form "Is $x \in D$?",*
3. *preparation of the quantum state $\frac{1}{\sqrt{\sum_{x\in D} c_x}} \sum_{x\in D} \sqrt{c_x}\,|x\rangle$.*

## 3 USTCON and the Quantum Walk model

In this section we define the undirected *st*-connectivity problem (USTCON). The input to this problem is an undirected graph $G = (X, E)$. Classically, there are various ways this input may be given, which may change the complexity of the problem. For example, in the *adjacency array model* (defined below), it is possible to randomly sample a neighbour of any vertex $u$ in $O(1)$ queries to $G$ (assuming access to the vertex degrees), facilitating a random walk on $G$, whereas if $G$ is given as an *adjacency matrix*, a random walk step is not so simple.

We will work in a quantum walk analogue of the adjacency array model. We assume that $G$ can be accessed via the *quantum walk oracle* that for every $u \in X$ outputs a uniform superposition over its neighbours:[3]

$$\mathcal{O}_W : |u\rangle\,|0\rangle \mapsto \frac{1}{\sqrt{d_u}} \sum_{v\in N(u)} |u\rangle\,|v\rangle\,. \tag{2}$$

Formally, we describe $\mathrm{USTCON}_{\mathrm{qw}}$ in terms of the input and output.

▶ **Problem 6** ($\mathrm{USTCON}_{\mathsf{qw}}$). *Given access to an undirected graph $G = (X, E)$ via the quantum walk oracle $\mathcal{O}_W$, and two vertices $s, t \in X$, decide whether $s$ and $t$ are in the same connected component of $G$.*

To compare our work with classical results on $\mathrm{USTCON}_{\mathrm{arr}}$, we describe an implementation of the quantum walk oracle defined above based on adjacency array access to a graph. Let $u \in X$ and $i \in [d_u]$. We assume that for each vertex $u$ there is a fixed numbering of its neighbours from 1 to $d_u$. In the *adjacency array model*, two types of queries are allowed:

- Degree query $\mathcal{O}_D : |0\rangle\,|u\rangle \mapsto |d_u\rangle\,|u\rangle$
- Neighbour query $\mathcal{O}_N : |u\rangle\,|i\rangle\,|0\rangle \mapsto |u\rangle\,|i\rangle\,|v_i(u)\rangle$

In the *sorted adjacency array model* we additionally assume that for every vertex $u \in X$ its neighbours are sorted: for any $i, j \in [d_u]$, if $i < j$ then $v_i(u) < v_j(u)$. In particular, this allows us to check with $O(\log(n))$ queries whether a given pair of vertices $u, v$ are adjacent. We define the USTCON-problem in this model as follows.

▶ **Problem 7** ($\mathrm{USTCON}_{\mathsf{s\text{-}arr}}$). *Given access to an undirected graph $G = (X, E)$ via the sorted adjacency array model and two vertices $s, t \in X$, decide whether $s$ and $t$ are in the same connected component of $G$.*

The question that we consider is how many sorted adjacency array queries to the graph it takes to implement the quantum walk oracle $\mathcal{O}_W$.

[3] Note that this is exactly the quantum walk oracle defined in Theorem 4, specialized to unweighted graphs.

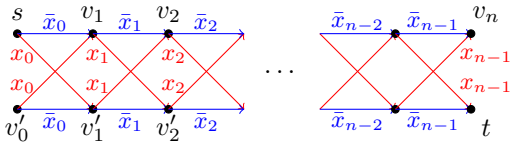



**Figure 2** The parity graph. We include an edge labelled by "$x_i$" (in red) if and only if $x_i = 1$, and an edge labelled "$\bar{x}_i$" (in blue) if and only if $x_i = 0$, meaning that for each vertex we include exactly one of the two incoming edges, and exactly one of the two outgoing edges. The resulting graph has $s$ and $t$ connected if and only if $\text{PARITY}(x) = 1$.

▶ **Lemma 8.** *The quantum walk oracle $\mathcal{O}_W$ for an unweighted graph $G$ (Equation (2)) with maximum degree $d_{\max}$ can be implemented with $O(\log(d_{\max}))$ queries in the sorted adjacency array model, and $\widetilde{O}(1)$ other elementary operations and space.*

The proof of this lemma is deferred to the appendix.

It follows that any quantum algorithm solving $\text{USTCON}_{\text{qw}}$ in $T$ time and $S = \Omega(\log(n))$ space can solve $\text{USTCON}_{\text{s-arr}}$ in $\widetilde{O}(T)$ time and $O(S)$ space.

## 4 Time- and space-optimal quantum algorithm

In Section 4.2 and Section 4.3, we give an algorithm for $\text{USTCON}_{\text{qw}}$ that is optimal in both time and space. For completeness, we first give a time lower bound in Section 4.1.

### 4.1 Lower bound

The proof of the following lower bound follows the lines of the proof of an analogous lower bound for the strong connectivity problem described in [13]. The proof is via a reduction from PARITY.

▶ **Problem 9** (PARITY). *Given oracle access to a string $x \in \{0,1\}^n$ via $\mathcal{O}_x : |i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$, return $\bigoplus_{i=0}^{n-1} x_i$.*

▶ **Lemma 10** ([6, 15]). *The bounded error quantum query complexity of PARITY is $\Omega(n)$.*

We use Lemma 10 and a reduction from parity, using the parity graph illustrated in Figure 2, to show the following result. The detailed proof can be found in the appendix.

▶ **Theorem 11.** *The bounded error quantum query complexity of $\text{USTCON}_{\text{s-arr}}$ and $\text{USTCON}_{\text{qw}}$ is $\Omega(n)$.*

### 4.2 Metropolis-Hastings walk

In this section, we consider an unweighted simple graph $G$. The algorithm that we propose involves a quantum walk on a modified weighted version of $G$ that we call $G' = (X', E', W)$. We start by describing the construction of $G'$ that was introduced in [19, arXiv v2].

▶ **Definition 12** (Metropolis-Hastings walk). *For any graph $G = (X, E)$, the corresponding* Metropolis-Hastings walk *is the random walk on the weighted graph $G' = (X', E', W)$ defined as follows. For every $u \in X$, we include a corresponding vertex $x_u$ in $X'$. In addition, for every edge $\{u, v\} \in E$, we add a new vertex $x_{u,v}$ that splits the edge into two new edges. Formally:*

$$X' = \{x_u : u \in X\} \cup \{x_{u,v} : \{u,v\} \in E, u < v\}$$
$$E' = \{\{x_u, x_{u,v}\} : \{u,v\} \in E\}.$$

*For every edge $\{x_u, x_{u,v}\} \in E'$, we define the edge weight $W_{x_u,x_{u,v}} = \frac{1}{d_u}$. These weights define transition probabilities for the random walk on $G'$.*

The above has been called the *cautious walk* in [19, arXiv v2], while *Metropolis-Hastings-type* walks are walks in which neighbours are sampled and accepted with some probability. Our terminology is motivated by the following observation. If we start with a vertex $u \in X$ and take two steps of the walk of Definition 12, then we arrive at another vertex $v \in X$, which is either the same or a neighbour of $u$ in $G$. The walk on $G$ defined this way has the following alternative description: sample a uniformly random neighbour and accept it with probability $\frac{1/d_v}{1/d_u+1/d_v} = \frac{1}{1+d_v/d_u}$. This is precisely a random walk that falls into the Metropolis-Hastings framework, justifying our terminology. The precise choice of acceptance probabilities is sometimes called the *Glauber choice* in the literature (e.g., [20]). We note that a later version of [19] uses another choice of Metropolis-Hastings walk, but of our purposes we find it convenient to stick to the walk as defined above.

While the hitting time of a random walk between two vertices in $G$ may be as high as $O(n^3)$, in $G'$ it is at most $O(n^2)$ [19, Lemma 2 of arXiv v2]:

▶ **Lemma 13** ([19]). *Let $G = (X, E)$ be any unweighted graph, and $G'$ the corresponding (weighted) Metropolis-Hastings graph as in Definition 12. For any $u, v \in X$ connected by a path, $\mathcal{H}_{u,v}(G') \leq 18n^2$.*

In order to apply Theorem 4 to $G'$, we need to upper bound $\mathsf{U}$, the cost of implementing the weighted quantum walk oracle. For $u \in X'$ the oracle is defined as

$$U : |x\rangle |0\rangle \mapsto \sum_{y \in N(x)} \sqrt{P'_{xy}} |x\rangle |y\rangle , \quad \forall x \in X'$$

where $P'_{xy}$ is the probability of walking from $x$ to $y$ defined by the edge weights.

▶ **Lemma 14.** *The weighted quantum walk oracle $U$ for the Metropolis-Hastings walk $G'$ can be implemented with $\widetilde{O}(1)$ degree queries $\mathcal{O}_D$, $\widetilde{O}(1)$ applications of the quantum walk operator $\mathcal{O}_W$ on the graph $G$, and $\widetilde{O}(1)$ additional gates.*

*Therefore, $U$ can be implemented with $\widetilde{O}(1)$ queries to $G$ in the sorted adjacency array model, and $\widetilde{O}(1)$ additional gates.*

**Proof.** Note that the first statement implies the second one due to Lemma 8. Therefore, we will only prove the first statement. Consider the following encoding of the vertices of $G'$.

$$X' = \{(u,0) : u \in X\} \cup \{(u,v) : \{u,v\} \in E, u < v\} \subseteq X \times \big(X \cup \{0\}\big),$$

where 0 is a null symbol not contained in $X$. The first set of this union corresponds to original vertices of $G$ and the second one corresponds to the added ones.

To implement $U$ on $|x\rangle |0\rangle$, we first compute a bit in an ancilla register $A$ that is $|0\rangle_A$ if $x = (u,0)$ for some $u$, and $|1\rangle_A$ otherwise. We will condition on this value.

First, conditioned on $|0\rangle_A$, our implementation proceeds as follows, for $u \in X$:

$$|x_u\rangle |0\rangle = |u,0\rangle |0\rangle \overset{J\mathcal{O}_W}{\mapsto} \frac{1}{\sqrt{d_u}} \sum_{v \in N(u)} |u\rangle |v\rangle |u,v\rangle$$
$$\overset{J'}{\mapsto} \frac{1}{\sqrt{d_u}} \sum_{v \in N(u)} |u,0\rangle |u,v\rangle = \frac{1}{\sqrt{d_u}} \sum_{v \in N(u)} |x_u\rangle |x_{u,v}\rangle = U |x_u\rangle |0\rangle$$

where $J$ is a unitary that acts as $|u,v\rangle|0\rangle \mapsto |u,v\rangle|u,v\rangle$, and $J'$ as $|u,v\rangle|u,v\rangle \mapsto |u,0\rangle|u,v\rangle$, each of which can be implemented with $O(\log(n))$ controlled-NOT gates (since every vertex is described by $O(\log(n))$ bits).

Next, conditioned on $|1\rangle_A$, our implementation proceeds as follows, for $\{u,v\} \in E$ with $u < v$, and $|0\rangle_{A'}$ a fresh ancilla:

$$
\begin{aligned}
|0\rangle_{A'}|x_{u,v}\rangle|0\rangle &= |0\rangle_{A'}|u,v\rangle|0\rangle \\
&\overset{1}{\mapsto} \left(\sqrt{\frac{1/d_u}{1/d_u+1/d_v}}|0\rangle + \sqrt{\frac{1/d_v}{1/d_u+1/d_v}}|1\rangle\right)_{A'}|u,v\rangle|0\rangle \\
&\overset{2}{\mapsto} \sqrt{\frac{1/d_u}{1/d_u+1/d_v}}|0\rangle_{A'}|u,v\rangle|u\rangle + \sqrt{\frac{1/d_v}{1/d_u+1/d_v}}|1\rangle_{A'}|u,v\rangle|v\rangle \\
&\overset{3}{\mapsto} |0\rangle_{A'}\left(\sqrt{\frac{1/d_u}{1/d_u+1/d_v}}|u,v\rangle|u\rangle + \sqrt{\frac{1/d_v}{1/d_u+1/d_v}}|u,v\rangle|v\rangle\right) \\
&= |0\rangle_{A'}\left(\sqrt{\frac{W_{x_{u,v},x_u}}{w(x_{u,v})}}|x_{u,v}\rangle|x_u\rangle + \sqrt{\frac{W_{x_{u,v},x_v}}{w(x_{u,v})}}|x_{u,v}\rangle|x_v\rangle\right) \\
&= |0\rangle_{A'}U|x_{u,v}\rangle|0\rangle,
\end{aligned}
$$

where we use the following mappings:

**1:** Query degrees for $u$ and $v$ into a new ancilla register, perform the rotation controlled on the degrees (cf. [17]), and then uncompute the degrees ($O(1)$ degree queries to $G$).

**2:** Controlled on the first register, select one of the two vertices to copy into the last register ($O(\log(n))$ Toffoli gates).

**3:** Flip the bit in $A'$ if the second vertex of $|u,v\rangle$ is the same as the one written in the third register ($O(\log(n))$ elementary gates).

To complete the proof, note that we can uncompute the bit in ancilla $A$, because the register containing $|x\rangle$ has not been changed. ◀

## 4.3 The algorithm

We can solve $\text{USTCON}_{\text{qw}}(G)$ using Algorithm 1. This leads to our main theorem of this section.

**Algorithm 1** Quantum algorithm for $\text{USTCON}_{\text{qw}}$ with optimal time and space.

> Apply the algorithm from Theorem 4 to the Metropolis-Hastings walk $P'$ with $M = \{t\}$, using Lemma 14 to implement the quantum walk oracle for $G'$. If the algorithm returns $t$, output "connected", and otherwise output "disconnected".

▶ **Theorem 15.** *There exists a $O(\log(n))$-space quantum algorithm that decides $\text{USTCON}_{\text{qw}}$ and $\text{USTCON}_{\text{s-arr}}$ with bounded one-sided error in $\widetilde{O}(n)$ gates and queries.*

**Proof.** Let $X_s \subseteq X$ denote the connected component of $s$. If $t \in X_s$, the algorithm will output $t$ with probability at least $2/3$, in which case our algorithm will output the correct answer, "connected". If $t \notin X_s$, then the algorithm will output an element of $X_s$ with probability 1, in which case, our algorithm will output the correct answer "disconnected". This establishes correctness of Algorithm 1 with one-sided error.

To analyze the complexity, note that by Lemma 14 we have $\mathsf{U} = \widetilde{O}(1)$. For any $u \in X_s$, we can check if $u \in M$ by checking if $u = t$ in complexity $\mathsf{C} = O(\log(n)) = \widetilde{O}(1)$. To complete the analysis, we need only upper bound the commute time between $s$ and $t$ when $t \in X_s$. Since $\mathcal{C}_{s,t} = \mathcal{H}_{s,t} + \mathcal{H}_{t,s}$, by Lemma 13, we have $\mathcal{C}_{s,t} \leq 36n^2 =: \mathcal{C}$. Thus, referring to Theorem 4, the complexity of our algorithm is:

$$\widetilde{O}\left(\sqrt{\mathcal{C}\log(\mathcal{C})\log(\log(\mathcal{C}))}(\mathsf{C}+\mathsf{U})\right) = \widetilde{O}(n). \qquad \blacktriangleleft$$

## 5 Time-space tradeoff for bounded spectral gap

In this section we revisit the problem of undirected *st*-connectivity in the setting where one is given a lower bound on the spectral gap of the random walk. As discussed in Section 2.2, such a bound is tightly related to the mixing time of the walk. We will give a quantum algorithm that exhibits a nontrivial time-space tradeoff in this setting.

Our discussion will be general and apply to random walks on weighted graphs as defined in Equation (1). This is useful since the spectral gaps and mixing times of random walks on $G$ with different edge weights are in general *not* comparable. E.g., on the lollipop graph (an $n$-vertex clique connected to an $O(n)$-vertex path) the mixing time of the unweighted random walk is $\Theta(n^3)$ [10], while it is $O(n^2)$ for the Metropolis-Hastings walk.[4] On the other hand, on an $n$-vertex star graph the unweighted random walk has mixing time $O(1)$ while the Metropolis-Hastings walk has mixing time $\Theta(n)$. Thus, while the specific edge weights do not affect whether $s$ and $t$ are connected, they do impact the algorithm. Throughout this section, we assume some fixed edge weights are given, and we do not try to optimize for "good" edge weights. More specifically, we assume access to a *weighted quantum walk oracle* that for every vertex outputs a superposition of its neighbours, with squared amplitudes proportional to the edge weights:

$$\mathcal{O}_W : |u\rangle|0\rangle \mapsto \sum_{v\in N(u)} \sqrt{\frac{W_{u,v}}{w_u}}|u\rangle|v\rangle = \sum_{v\in N(u)} \sqrt{P_{u,v}}|u\rangle|v\rangle \qquad \forall u \in X$$

Moreover, we assume access to the weighted vertex degrees $w_u$ and that these degrees are of bounded absolute value for some $\mathrm{poly}(n)$ bound. This will allow us to generate the state $|\pi_{X'}\rangle$ for any subset $X' \subseteq X$ stored in QCRAM.

▶ **Problem 16** ($\textsc{ustcon}_{\mathsf{qw},\delta}$). *Given access to an undirected weighted graph via the quantum walk oracle $\mathcal{O}_W$, two vertices $s, t \in X$, and the promise that either $s$ and $t$ are disconnected or the spectral gap of the transition matrix of the walk is at least some $\delta > 0$, decide which is the case.*

Our main result of this section is the following:

▶ **Theorem 17.** *Fix $\delta \geq 0$. Let $\mathcal{G}_n$ be a family of undirected weighted graphs $G = (X, E, W)$ with $n = |X|$, such that $\gamma_\star(G) \geq \delta$ whenever $s$ and $t$ are connected. Then for any $S = \Omega(\log(n))$, there is a quantum algorithm that decides $\textsc{ustcon}_{\mathrm{qw},\delta}$ on $\mathcal{G}_n$ with bounded error in $O(S)$ space – of which $O(\log(n/\delta))$ is quantum memory, and the remainder is QCRAM – and $T = \widetilde{O}(\frac{S}{\delta}\log\left(\frac{1}{\pi_{\min}}\right) + \sqrt{\frac{n}{\delta S}})$ queries to $\mathcal{O}_W$, elementary gates, and QCRAM queries.*

[4] This follows from the $O(n^2)$ upper bound on the maximum hitting time of the Metropolis-Hastings walk (Lemma 13), and the fact that the maximum hitting time upper bounds the mixing time [21, Lemma 10.2].

Note that we can assume $\delta \geq 1/n$. If $\delta < 1/n$ then $T \approx S/\delta$. There is no time-space tradeoff, and it is always faster to run the Metropolis-Hastings algorithm (Algorithm 1).

The algorithm is stated below as Algorithm 2. It consists of three stages. We fix some parameter $p$, which denotes the number of "pebbles", or vertices the algorithm will keep track of (so $S = O(p \log(n))$). First, we run $O(p)$ classical random walks starting from $s$, each of length $\ell = O\left(\frac{1}{\delta} \log\left(\frac{n}{p\pi_{\min}}\right)\right)$. This allows us to sample a set $L$ of $O(p)$ points from $X_s$, the connected component of $s$ (the big-O notation suppresses a universal constant that is given in the proof). Since $\ell$ is at least the mixing time of $G$ (see Theorem 3), assuming $s$ and $t$ are connected, each point is sampled (approximately) from $\pi$. We do the same from $t$ to get a random subset $M \subseteq X_t$ connected to $t$.

Next, we use $L$ and $M$ to prepare (up to some error) the states $|\pi_{X_s}\rangle$ and $|\pi_{X_t}\rangle$, using *inverse quantum walk search*, which we describe in more detail in Section 5.2. If $s$ and $t$ are in the same connected component, then $|\pi_{X_s}\rangle = |\pi_{X_t}\rangle$, and otherwise, the states are orthogonal. The final step is to distinguish these two cases using a SWAP test [1, Claim 1]. This roughly follows an earlier approach in [3], the main difference being that we sample the sets $L$ and $M$ using a random walk (which allows us to exploit the gap promise), while in [3] the sets are constructed using a breadth-first search.

**Algorithm 2** Quantum algorithm for $\text{USTCON}_{\text{qw}}$ with a tradeoff.

`Seed set:` Run $O(p)$ classical random walks from $s$ and $O(p)$ classical random walks from $t$, each for $O(\frac{1}{\delta} \log\left(\frac{n}{p\pi_{\min}}\right))$ steps. Let $L$ and $M$ denote the respective sets of endpoints, without duplicates. If $L \cap M \neq \emptyset$, return "connected".[5]

`State preparation:` Run inverse quantum walk search from $|\pi_L\rangle$ and $|\pi_M\rangle$ for time $\widetilde{O}\left(\sqrt{\frac{n}{\delta p}}\right)$ to prepare $|\pi_{X_s}\rangle$ and $|\pi_{X_t}\rangle$, respectively, to precision $1/8$.

`SWAP test:` Do a SWAP test on the resulting states. If the test returns "0", return "connected", otherwise return "disconnected".

If we specialize Theorem 17 to the unweighted graph case, we get the following corollary.

▶ **Corollary 18.** *For any $S > 0$, there is a quantum algorithm that solves $\text{USTCON}_{\text{s-arr}}$ with a promise $\delta > 0$ on the random walk spectral gap using space $S$ and time $T \in \widetilde{O}(S/\delta + \sqrt{n/(\delta S)})$.*

An analogous result holds for the Metropolis-Hastings walk described in Section 4.2 given a promise on its spectral gap, since we showed that the corresponding quantum walk oracle can also be efficiently implemented.

In the remainder of this section we will analyze each stage of Algorithm 2.

## 5.1 Analysis of step 1: Seed set

Recall that the first stage results in random sets $L = \{x_1, \ldots, x_{cp}\}$ and $M = \{y_1, \ldots, y_{cp}\}$, where $x_1, \ldots, y_{cp}$ are the endpoints of independent random walks starting at $s$ or $t$, respectively, and $c > 0$ is some universal constant that we will choose later. Since we run those random walks for $O(\frac{1}{\delta} \log\left(\frac{n}{p\pi_{\min}}\right))$ steps, by Theorem 3 it follows that the $x_j$ are independent

[5] We use the following strategy to check whether $L \cap M \neq \emptyset$: Sort $L$ and $M$ and check for every element of $L$ if it is present in $M$. This takes $\widetilde{O}(p)$ time and space.

samples drawn from a distribution $\tilde{\pi}$ such that

$$\|\tilde{\pi} - \pi\|_{\mathrm{TV}} \leq \frac{p}{8n},$$

where $\pi$ is the stationary distribution on $X_s$. If $s$ and $t$ are connected then $X_s = X_t$ and the samples $y_j$ are similarly drawn from a distribution that is $p/(8n)$-close to $\pi$. Here we prove that this implies lower bounds on the stationary measure of the sets $L$ and $M$.

▶ **Proposition 19.** *There exists a universal constant $c > 0$ such that the following holds. Let $p \in [n]$ and assume $L \subseteq X$ is a random set obtained by sampling $cp$ independent elements from a distribution $\tilde{\pi}$ such that $\|\tilde{\pi} - \pi\|_{\mathrm{TV}} \leq \frac{p}{8n}$ (and removing duplicates). Then, $\mathbf{Pr}(\pi(L) \geq \frac{p}{8n}) \geq \frac{9}{10}$.*

The proof of the proposition uses the following lemma, which formalizes the intuition that adding a random element to a low-probability subset should increase the probability.

▶ **Lemma 20.** *Let $X$ be a set of cardinality $n$, $A \subseteq X$ an arbitrary fixed subset, and let $b$ be drawn at random from an arbitrary distribution $\sigma$ on $X$. Then:*

$$\mathbf{Pr}\left(\sigma(A \cup \{b\}) > \sigma(A) + \frac{1 - \sigma(A)}{2n}\right) \geq \frac{1 - \sigma(A)}{2}.$$

**Proof.** Say $b$ is *bad* if $b \in A$ or $\sigma(b) \leq \frac{1-\sigma(A)}{2n}$, and *good* otherwise. Then

$$\mathbf{Pr}\left(\sigma(A \cup \{b\}) > \sigma(A) + \frac{1 - \sigma(A)}{2n}\right) = \mathbf{Pr}(b \text{ is good}) = 1 - \mathbf{Pr}(b \text{ is bad}).$$

We can compute:

$$\begin{aligned}\mathbf{Pr}(b \text{ is bad}) &= \sigma\left(A \cup \left\{x \in X : \sigma(x) \leq \frac{1 - \sigma(A)}{2n}\right\}\right) \\ &\leq \sigma(A) + n \cdot \frac{1 - \sigma(A)}{2n} = \frac{1 + \sigma(A)}{2},\end{aligned}$$

from which the claim follows. ◀

**Proof of Proposition 19.** Let $x_1, x_2, \ldots$ denote samples drawn independently at random from $\tilde{\pi}$. For any integer $T \geq 1$, define $L_T := \{x_1, \ldots, x_T\}$ as the set consisting of the first $T$ samples (with duplicates removed), as well as $L_0 := \emptyset$. We say that the $T$-th sample is a *success* if

$$\tilde{\pi}(L_{T-1}) \geq \frac{1}{2} \quad \text{or} \quad \tilde{\pi}(L_T) \geq \tilde{\pi}(L_{T-1}) + \frac{1}{4n},$$

and a *failure* otherwise. Let $T_j$ denote the index of the $j$-th success, with $T_0 := 0$. Then, clearly,

$$\tilde{\pi}(L_{T_p}) \geq \min\left\{\frac{1}{2}, \frac{p}{4n}\right\} = \frac{p}{4n} \quad \text{and hence} \quad \pi(L_{T_p}) \geq \tilde{\pi}(L_{T_p}) - \frac{p}{8n} \geq \frac{p}{8n}.$$

On the other hand, note that by Lemma 20, the $T$-th sample is a success with probability at least $\frac{1}{4}$, even if we condition on all prior samples. In particular, the probability that there are $k$ failures in a row is at most $(\frac{3}{4})^k$. Therefore,

$$\mathbf{E}[T_j - T_{j-1}] = \sum_{k=1}^{\infty} k \,\mathbf{Pr}(T_j = T_{j-1} + k) \leq \sum_{k=1}^{\infty} k \left(\frac{3}{4}\right)^k = 12$$

and hence, by Markov's inequality,

$$\mathbf{Pr}(T_p > cp) \leq \frac{1}{cp}\,\mathbf{E}[T_p] = \frac{1}{cp}\sum_{j=1}^{p}\mathbf{E}[T_j - T_{j-1}] \leq \frac{12p}{cp} = \frac{1}{10}$$

provided we choose $c := 120$. Since the random set $L$ in the statement of the lemma is defined by taking $cp$ many samples, we obtain that

$$\mathbf{Pr}\Big(\pi(L) \geq \frac{p}{8n}\Big) \geq \mathbf{Pr}(T_p \leq cp) \geq 1 - \frac{1}{10} = \frac{9}{10}.$$ ◀

## 5.2 Analysis of step 2: State preparation

Now we turn to the analysis of the quantum walk search routine in step 2 of the algorithm. We rely on the following proposition from [3], which formalizes the idea of "inverse quantum walk search".[6]

▶ **Proposition 21** ([3, Proposition 1]). *Consider a subset $A \subseteq X$ of a (connected) graph $G$, and let $\delta$ be a lower bound on the spectral gap of a random walk $P$ on $G$ with stationary distribution $\pi$. From $|\pi_A\rangle$, we can generate a state $|\tilde{\pi}\rangle = |\pi\rangle + |\Gamma\rangle$ with $\| |\Gamma\rangle \|_2 \leq \epsilon$ using an expected number of calls to the weighted quantum walk oracle*

$$O\left(\frac{1}{\sqrt{\pi(A)\delta}}\log\left(\frac{1}{\pi(A)\epsilon}\right)\right),$$

*and $O(1/\sqrt{\pi(A)\delta})$ reflections around $|\pi_A\rangle$. The algorithm uses space logarithmic in $n$, $1/\delta$, $1/\pi(A)$ and $1/\epsilon$.*

The proposition implies the following.

▷ Claim 22. Step 2 of Algorithm 2 prepares 1/8-approximations of $|\pi_{X_s}\rangle$ and $|\pi_{X_t}\rangle$ with probability at least 9/10 in time complexity $O\left(\sqrt{\frac{n}{p\delta}}\log\left(\frac{n}{p}\right)\right)$.

## 5.3 Analysis of step 3: SWAP test

In the last step of our algorithm, we wish to decide whether $|\pi_{X_s}\rangle = |\pi_{X_t}\rangle$ or whether they are orthogonal. For this we use the SWAP test.

▷ Claim 23. Step 3 of Algorithm 2 decides whether $|\pi_{X_s}\rangle = |\pi_{X_t}\rangle$ or whether they are orthogonal with constant probability in time $\widetilde{O}(1)$.

Proof. Using a single copy of two states $|\psi\rangle$ and $|\psi'\rangle$, and $O(\log(n))$ additional gates, the SWAP test returns "0" with probability $(1 + |\langle\psi|\psi'\rangle|^2)/2$ and "1" with probability $(1 - |\langle\psi|\psi'\rangle|^2)/2$ [1, Claim 1].

By Claim 22, in step 2 we prepared states $|\tilde{\pi}_{X_s}\rangle = |\pi_{X_s}\rangle + |\Gamma_L\rangle$ and $|\tilde{\pi}_{X_t}\rangle = |\pi_{X_t}\rangle + |\Gamma_M\rangle$ such that $\| |\Gamma_L\rangle \|_2, \| |\Gamma_M\rangle \|_2 \leq 1/8$ with probability 9/10. By a triangle inequality, this implies that

$$\Big|\,|\langle\tilde{\pi}_{X_s}|\tilde{\pi}_{X_t}\rangle| - |\langle\pi_{X_s}|\pi_{X_t}\rangle|\,\Big| < 1/3,$$

and so $|\langle\tilde{\pi}_{X_s}|\tilde{\pi}_{X_t}\rangle| > 2/3$ if $s$ and $t$ are connected, but $|\langle\tilde{\pi}_{X_s}|\tilde{\pi}_{X_t}\rangle| < 1/3$ otherwise. The SWAP test distinguishes these cases with constant probability. ◁

[6] [3, Proposition 1] only proves the proposition for simple random walks, however it trivially extends to random walks on weighted graphs.

## 5.4 Proof of Theorem 17

In this section, we prove Theorem 17, which we restate here for convenience.

▶ **Theorem 17.** *Fix $\delta \geq 0$. Let $\mathcal{G}_n$ be a family of undirected weighted graphs $G = (X, E, W)$ with $n = |X|$, such that $\gamma_\star(G) \geq \delta$ whenever $s$ and $t$ are connected. Then for any $S = \Omega(\log(n))$, there is a quantum algorithm that decides $\text{USTCON}_{\text{qw},\delta}$ on $\mathcal{G}_n$ with bounded error in $O(S)$ space – of which $O(\log(n/\delta))$ is quantum memory, and the remainder is QCRAM – and $T = \widetilde{O}(\frac{S}{\delta}\log\left(\frac{1}{\pi_{\min}}\right) + \sqrt{\frac{n}{\delta S}})$ queries to $\mathcal{O}_W$, elementary gates, and QCRAM queries.*

**Proof.** We first analyze the space complexity of Algorithm 2. Step 1 is purely classical, and uses $O(p\log(n))$ space to store the $O(p)$ vertices in $L$ and $M$, with each random walk using $O(\log(n))$ space. We can implement step 2 using the algorithm referred to in Proposition 21 using $O(\log(n))$ qubits of space, but this requires that the $O(p\log(n))$ classical space used to store $L$ and $M$ in step 1 is QCRAM. Finally, step 3 just uses $O(\log(n))$ quantum space. Thus, the claimed space complexity follows if we set $S = p\log(n)$.

Next, we analyze the time complexity. Every random walk of step 1 adds $O\left(\frac{1}{\delta}\log\left(\frac{n}{p\pi_{\min}}\right)\right)$ to the time complexity. The total number of walks is $O(p)$. Checking whether $L \cap M = \emptyset$ is $O(p)$ as this is the total number of samples. Hence, the overall complexity of the first step is $\widetilde{O}\left(\frac{p}{\delta}\log\left(\frac{1}{\pi_{\min}}\right)\right)$. By Claim 22, the complexity of step 2 is $O\left(\sqrt{\frac{n}{p\delta}}\log\left(\frac{n}{p}\right)\right)$. Finally, the SWAP test in step 3 uses only $O(\log(n))$ gates, since the states being compared are $O(\log(n))$-qubit states. Hence, the total time complexity of Algorithm 2 is

$$T = \widetilde{O}\left(\frac{p}{\delta}\log\left(\frac{1}{\pi_{\min}}\right) + \sqrt{\frac{n}{\delta p}}\right) = \widetilde{O}\left(\frac{S}{\delta}\log\left(\frac{1}{\pi_{\min}}\right) + \sqrt{\frac{n}{\delta S}}\right),$$

since $S = \widetilde{O}(p)$.

Finally, for the correctness of the algorithm, by Claim 23, Algorithm 2 distinguishes between the case where $|\pi_{X_s}\rangle$ and $|\pi_{X_t}\rangle$ are equal and the case where they are orthogonal with bounded error. If $X_s = X_t$ (i.e. $s$ and $t$ are connected) then the states are equal, and if $X_s \cap X_t = \emptyset$ (i.e. $s$ and $t$ are not connected) then they are orthogonal. ◀

## References


1. Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 20–29, 2003.
2. Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovasz, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 218–223, 1979. doi:10.1109/SFCS.1979.34.
3. Simon Apers. Quantum walk sampling by growing seed sets. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA)*, volume 144, pages 9:1–9:12. Springer, 2019.
4. Simon Apers, András Gilyén, and Stacey Jeffery. A unified framework of quantum walk search. In *Proceedings of the 38th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 6:1–6:13, 2020. arXiv:1912.04233 doi:10.4230/LIPIcs.STACS.2021.6.
5. Greg Barnes and Uriel Feige. Short random walks on graphs. In *Proceedings of the 1993 ACM Symposium on the Theory of Computing (STOC)*, pages 728–737, 1993.
6. Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Earlier version in FOCS'98. arXiv:quant-ph/9802049 doi:10.1145/502090.502097.

7 Paul Beame, Allan Borodin, Prabhakar Raghavan, Walter L Ruzzo, and Martin Tompa. Time-space tradeoffs for undirected graph traversal. In *Proceedings of the 1990 IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 429–438. IEEE, 1990.

8 Aleksandrs Belovs. Quantum walks and electric networks. `arXiv:1302.3143`, 2013.

9 Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for *st*-connectivity and claw detection. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA)*, pages 193–204, 2012. `doi:10.1007/978-3-642-33090-2_18`.

10 Graham Brightwell and Peter Winkler. Maximum hitting time for random walks on graphs. *Random Structures & Algorithms*, 1(3):263–276, 1990.

11 Andrei Z Broder, Anna R Karlin, Prabhakar Raghavan, and Eli Upfal. Trading space for time in undirected st-connectivity. In *Proceedings of the 1989 ACM Symposium on the Theory of Computing (STOC)*, pages 543–549, 1989.

12 Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998.

13 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. Earlier version in ICALP'04. `arXiv:quant-ph/0401091 doi:10.1137/050644719`.

14 Jeff Edmonds. Time-space trade-offs for undirected st-connectivity on a JAG. In *Proceedings of the 1993 ACM Symposium on the Theory of Computing (STOC)*, pages 718–727, 1993.

15 Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Limit on the speed of quantum computation in determining parity. *Physical Review Letters*, 81(24):5442, 1998. `arXiv:quant-ph/9802045 doi:10.1103/PhysRevLett.81.5442`.

16 Uriel Feige. A randomized time-space tradeoff of $\tilde{O}(m\hat{R})$ for USTCON. In *Proceedings of the 1993 IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 238–246. IEEE, 1993.

17 Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*, 2002.

18 Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 49:1–49:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

19 Adrian Kosowski. Faster walks in graphs: A $\tilde{O}(n^2)$ time-space trade-off for undirected s-t connectivity. In *Proceedings of the 2013 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1873–1883, 2013. `arXiv:1204.1136v2 doi:10.1137/1.9781611973105.133`.

20 Jessica Lemieux, Bettina Heim, David Poulin, Krysta Svore, and Matthias Troyer. Efficient quantum walk circuits for metropolis-hastings algorithm. *Quantum*, 4:287, 2020.

21 David A Levin, Yuval Peres, and Elizabeth L Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2017. second edition.

22 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008. `doi:10.1145/1391289.1391291`.

23 Avi Wigderson. The complexity of graph connectivity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 112–132. Springer, 1992.


# A Appendix

**Proof of Lemma 8.** Assume first that there is an additional type of query allowed, namely:

$$\text{Index query: } \mathcal{O}_I : |u\rangle |v\rangle |0\rangle \mapsto |u\rangle |v\rangle |i\rangle \tag{3}$$

for $u, v \in X$ and $i \in [d_u]$ such that $v_i(u) = v$. Then $\mathcal{O}_W$ can be implemented using $\mathcal{O}_I$, $\mathcal{O}_D$, and $\mathcal{O}_N$ as follows. Let $F_d$ denote the Fourier transform over $\mathbb{Z}_d$, and let $F = \sum_{d=1}^{n} |d\rangle \langle d| \otimes F_d$, which can be implemented (to any inverse polynomial precision) in $O(\log(n))$ gates [12]. Then for any $u \in X$, we implement:

$$|0\rangle|u\rangle|0\rangle|0\rangle \overset{\mathcal{O}_D}{\mapsto} |d_u\rangle|u\rangle|0\rangle|0\rangle \overset{F}{\mapsto} \frac{1}{\sqrt{d_u}}|d_u\rangle\sum_{i=1}^{d_u}|u\rangle|i\rangle|0\rangle$$
$$\overset{\mathcal{O}_N}{\mapsto} \frac{1}{\sqrt{d_u}}|d_u\rangle\sum_{i=1}^{d_u}|u\rangle|i\rangle|v_i(u)\rangle \overset{\mathcal{O}_I^\dagger\mathcal{O}_D^\dagger}{\mapsto} \frac{1}{\sqrt{d_u}}\sum_{i=1}^{d_u}|u\rangle|v_i(u)\rangle = \mathcal{O}_W|u\rangle|0\rangle.$$

To complete the proof, note that the index query operator $\mathcal{O}_I$ only requires $O(\log(d_u))$ sorted adjacency array queries, since the neighbours are sorted and this makes it possible to perform binary search for $i$ such that $v_i(u) = v$. ◀

**Proof of Theorem 11.** We will reduce the PARITY problem to $\text{USTCON}_{\text{s-arr}}$. Since PARITY requires $\Omega(n)$ queries by Lemma 10, and the quantum walk oracle $\mathcal{O}_W$ can be implemented using $\widetilde{O}(1)$ sorted array queries by Lemma 8, this reduction will prove the statement of the theorem.

Let $x \in \{0,1\}^n$ be an input of PARITY. The corresponding output would be $\bigoplus_{i=0}^{n-1} x_i$. Given this, we need to build a $\text{USTCON}_{\text{s-arr}}$ input that can be queried with a constant number of queries to $x$. Consider an undirected graph $G = (X, E)$ defined as follows (see also Figure 2):

$$X = \{s = v_0, v_0', v_1, v_1', \ldots, v_{n-1}, v_{n-1}', v_n, t = v_n'\}$$
$$E = \{\{v_i, v_{i+1}'\}, \{v_i', v_{i+1}\} : i \in \{0,\ldots,n-1\}, x_i = 1\}$$
$$\cup \{\{v_i, v_{i+1}\}, \{v_i', v_{i+1}'\} : i \in \{0,\ldots,n-1\}, x_i = 0\}.$$

In this setting, $\bigoplus_{i=0}^{n-1} x_i = 1$ if and only if $s$ and $t$ are connected in the graph $G$.

Next, we describe how to implement queries $\mathcal{O}_D$ and $\mathcal{O}_N$ to $G$ as required by the $\text{USTCON}_{\text{s-arr}}$ problem, using queries to $\mathcal{O}_x$. Consider the following encoding of vertices of $G$. For $i \in \{0,\ldots,n\}$, we let $v_i = (i, 0)$, and $v_i' = (i, 1)$. That is, for a vertex $(i, b)$, $i \in \{0,\ldots,n\}$ encodes the "column" and $b \in \{0, 1\}$ encodes the "row". Assume that the vertices are ordered lexicographically, i.e.

$$(i, b_i) < (j, b_j) \iff i < j \text{ or } i = j, b_i < b_j.$$

Queries to $G$ are described according to this ordering.

1. Degree queries, $\mathcal{O}_D$, are trivial in this case as $d_{v_0} = d_{v_0'} = d_{v_n} = d_{v_n'} = 1$, and all other degrees are 2.
2. Since every vertex has degree at most 2, we explicitly describe neighbour queries, $\mathcal{O}_N$ for indices 1 and 2 such that the ordering assumption holds:
   - $\mathcal{O}_N : |i\rangle|b\rangle|1\rangle|0\rangle|0\rangle \mapsto |i\rangle|b\rangle|1\rangle|i-1\rangle|b\rangle \overset{\mathcal{O}_x}{\mapsto} |i\rangle|b\rangle|1\rangle|i-1\rangle|b \oplus x_{i-1}\rangle,\ \forall\, 0 < i \le n,$
   - $\mathcal{O}_N : |i\rangle|b\rangle|2\rangle|0\rangle|0\rangle \mapsto |i\rangle|b\rangle|2\rangle|i+1\rangle|b\rangle \overset{\mathcal{O}_x}{\mapsto} |i\rangle|b\rangle|2\rangle|i+1\rangle|b \oplus x_i\rangle,\ \forall\, 0 \le i < n.$

It can be seen from the formulas that queries to $G$ can be implemented with a constant number of queries to the parity input $x$. This implies the $\Omega(n)$ lower bound in $\text{USTCON}_{\text{s-arr}}$.

For $\text{USTCON}_{\text{qw}}$, note that the graph has bounded degree, and so by Lemma 8 we can simulate a query in this model using $O(1)$ queries in the sorted adjacency array model. This implies a similar $\Omega(n)$ lower bound for this model. ◀

# Exploration of Graphs with Excluded Minors

**Júlia Baligács**
Technische Universität Darmstadt, Germany

**Yann Disser**
Technische Universität Darmstadt, Germany

**Irene Heinrich**
Technische Universität Darmstadt, Germany

**Pascal Schweitzer**
Technische Universität Darmstadt, Germany

## Abstract

We study the online graph exploration problem proposed by Kalyanasundaram and Pruhs (1994) and prove a constant competitive ratio on minor-free graphs. This result encompasses and significantly extends the graph classes that were previously known to admit a constant competitive ratio. The main ingredient of our proof is that we find a connection between the performance of the particular exploration algorithm Blocking and the existence of light spanners. Conversely, we exploit this connection to construct light spanners of bounded genus graphs. In particular, we achieve a lightness that improves on the best known upper bound for genus $g \geq 1$ and recovers the known tight bound for the planar case ($g = 0$).

**2012 ACM Subject Classification** Theory of computation → Online algorithms; Theory of computation → Sparsification and spanners; Mathematics of computing → Graphs and surfaces

**Keywords and phrases** online algorithms, competitive analysis, graph exploration, graph spanners, minor-free graphs, bounded genus graphs

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.11

**Related Version** *Full Version*: http://arxiv.org/abs/2308.06823

## 1 Introduction

We study a classic online graph exploration problem that was first proposed by Kalyanasundaram and Pruhs in 1994 [29]. In this setting, a single agent needs to systematically traverse an initially unknown, undirected, connected graph with non-negative edge weights. Upon visiting a new vertex, the agent learns the unique identifiers of all adjacent vertices and the weights of the corresponding edges. The cost incurred when traversing an edge is simply its weight. The objective in online graph exploration is to visit all vertices of the graph and return to the starting vertex while minimizing the total cost.

The performance of a (deterministic) online algorithm Alg is measured in terms of competitive analysis. That is, given a graph $G$ and starting vertex $v$ of $G$, we compare the cost $\textsc{Alg}(G, v)$ of the traversal it produces to the cost of an offline optimum traversal $\textsc{Opt}(G)$. Note that the optimum cost corresponds to the length of a shortest TSP tour of $G$ and does not depend on $v$. We say that Alg is *(strictly) $\rho$-competitive* for a class of graphs if $\textsc{Alg}(G, v) \leq \rho \cdot \textsc{Opt}(G)$ for every graph $G$ in the class and every vertex $v$ of $G$. The *(strict) competitive ratio* of an algorithm Alg is given by $\inf\{\rho : \textsc{Alg} \text{ is } \rho\text{-competitive}\}$.

Kalyanasundaram and Pruhs [29] posed the following question: *Is there a deterministic algorithm for online graph exploration with a constant competitive ratio?* Several algorithms were proposed with a competitive ratio of $\mathcal{O}(\log(n))$ [31, 36], where $n$ is the number of vertices, but better competitive ratios are only known for restricted classes of graphs [29, 31, 33]. The best known lower bound on the competitive ratio is 10/3 [5]. In particular, the original question of Kalyanasundaram and Pruhs remains open.


We formalize a connection between the performance of the particular exploration algorithm Blocking and the existence of light spanners. Spanners were introduced in 1989 by Peleg and Schäffer [35] and have been instrumental in the development of approximation algorithms, particularly for TSP [3, 8, 9]. Here, a subgraph $H = (V, E_H)$ of a connected, undirected graph $G = (V, E)$ with edge weights $w\colon E \to \mathbb{R}_{\geq 0}$ is called a $(1+\varepsilon)$-*spanner* of $G$ if $d_H(u,v) \leq (1+\varepsilon)\, d_G(u,v)$ for all $u, v \in V$, where $d_H$ and $d_G$ denote the shortest-path distance in $H$ and $G$, respectively. Then, $H$ has *stretch* at most $(1+\varepsilon)$ and its *lightness* is $w(H)/w(\text{MST})$, where $w(H) := \sum_{e \in E_H} w(e)$ and MST denotes a minimum spanning tree of $G$.

We show that the online graph exploration algorithm Blocking has a constant competitive ratio on every class of graphs that admits spanners of constant lightness for a fixed stretch. Prominent graph classes with this property are the classes with a forbidden minor [9]. We thus, in particular, obtain a constant competitive ratio for online graph exploration on all graph classes excluding a minor. They encompass many other important classes, such as graphs of bounded genus or bounded treewidth. Overall, this result subsumes and significantly extends all previously known graph classes for which a competitive ratio of $o(\log(n))$ was known.

Regarding research for graph spanners, results typically revolve around the existence of good, in particular light, spanners. For example, the Erdős girth conjecture [19] is equivalent to a lower bound of $\Omega(n^{1/k})$ on the best lightness of a $(2k-1)$-spanner in unweighted graphs. While this conjecture remains unresolved, a nearly matching upper bound was proven by Chechik and Wulff-Nilsen [11]. Various constant upper bounds on the lightness are known for restricted classes of graphs [2, 9, 12, 24]. Our newly discovered connection to graph exploration also allows us to contribute an improved upper bound for graphs of bounded genus using the ideas given in [31].

**Our results.** We significantly expand the class of graphs on which the exploration problem admits a constant-competitive algorithm.

▶ **Theorem 1.** *For every graph $H$ and constant $\delta > 0$, there is a constant $c$ (depending on $H$ and $\delta$) such that Blocking$_\delta$ is $c$-competitive on $H$-minor-free graphs.*

The technical contribution of this work is a new-found connection between graph spanners and the performance of the exploration algorithm Blocking$_\delta$ (see Section 2.1) introduced by Megow et al. [31] based on an algorithm of Kalyanasundaram and Pruhs [29]. This connection will allow us to prove Theorem 1.

Prior to our work, the largest class of graphs which was known to admit a constant-competitive algorithm was the class of bounded genus graphs [31]. As an aside, we obtain a slightly stronger bound also for bounded genus graphs (cf. Corollary 13).

So far, Blocking$_\delta$ was only studied for constant choices of the parameter $\delta$, i.e., independent of the number of vertices $n$. It is known that its competitive ratio is at least $\Omega(n^{1/(4+\delta)})$ if $\delta$ is a constant [31]. This naturally raises the question of whether improvement is possible if $\delta$ may depend on $n$. We obtain the following results.

▶ **Theorem 2.** *Blocking$_{\log(n)}$ is $O(\log(n))$-competitive.*

This shows that Blocking$_{\log(n)}$ achieves the best previously known competitiveness. We complement this with the following lower bounds.

▶ **Theorem 3.** *The competitive ratio of Blocking$_\delta$, where $\delta = \delta(n) > 0$, is at least*

**a)** $\Omega(\log(n)/\log(\log(n)))$,

**b)** $\Omega(\log(n))$ *for* $\delta \in o(\log(n)/\log\log(n))$ *as well as for* $\delta \in \Omega(\log(n))$.

In particular, this shows that there is no $\delta$ such that BLOCKING$_\delta$ is constant-competitive, but it remains open, whether there is a choice of $\delta$ for which the algorithm is $o(\log(n))$-competitive.

Next, we exploit the connection between spanners and exploration in reverse to derive the existence of good spanners in bounded genus graphs.

▶ **Theorem 4.** *For every $\varepsilon > 0$, the greedy $(1+\varepsilon)$-spanner of a graph of genus $g$ has lightness at most $\left(1+\frac{2}{\varepsilon}\right)\left(1+\frac{2g}{1+\varepsilon}\right)$.*

Prior to our work, the best known bound was due to Grigni [24] who showed that every graph of genus $g \geq 1$ contains a $(1+\varepsilon)$-spanner of lightness $1+\frac{12g-4}{\varepsilon}$. Moreover, it is already known that planar graphs, i.e., graphs of genus 0, contain $(1+\varepsilon)$-spanners of lightness $1+\frac{2}{\varepsilon}$ and that this is best possible [2]. This means that Theorem 4 gives a tight bound in the case $g = 0$ and extrapolates this bound to graphs of larger genus.

**Related work.** Kalyanasundaram and Pruhs [29] introduced the online graph exploration problem and gave a constant-competitive algorithm for planar graphs. Megow, Mehlhorn and Schweitzer [31] revisited the algorithm, addressed some technical intricacies, and proposed their reinterpretation BLOCKING$_\delta$, which we also consider in this paper. They expanded the result by Kalyanasundaram and Pruhs and showed that the algorithm is constant-competitive on bounded genus graphs. Moreover, they suggested a new algorithm hDFS and showed that it is constant-competitive on graphs with a bounded number of different weights and $O(\log(n))$-competitive on general graphs.

Another very natural approach for exploration is the *Nearest Neighbor* algorithm, which, in each step, explores the unvisited vertex nearest to the current location. This algorithm has been studied extensively as a TSP heuristic. Rosenkrantz, Stearns and Lewis were able to show that its competitive ratio is $\Theta(\log(n))$ [36]. It turned out that the lower bound of $\Omega(\log(n))$ is already achieved on unweighted planar graphs [28] and on trees [23]. Eberle et al. [18] revisited the algorithm with learning augmentation.

In addition to planar and bounded genus graphs, the exploration problem has been studied on many more graph classes. For example, Miyazaki, Morimoto and Okabe were able to show that the competitive ratio of the exploration problem is $(1+\sqrt{3})/2$ on cycles and 2 on unweighted graphs. Other examples of such graph classes are tadpole graphs [10], unicyclic graphs [23], and cactus graphs [23].

Currently, the best known lower bound for the graph exploration problem is 10/3 which was shown by Birx, Disser, Hopp, and Karousatou [5]. Their construction builds on a previously known lower bound of 2.5 shown by Dobrev, Královič, and Markou [17]. Since the construction by Birx et al. is planar, the lower bound of 10/3 even holds when the problem is restricted to planar graphs.

Several other settings of the exploration problem have been studied, such as exploration on directed graphs [1, 13, 22, 21] or exploration with a team of agents [14, 15, 16]. Another problem which is closely related to graph exploration is online TSP, where a single agent has to serve requests appearing over time in a known graph [6, 7].

Through the connection with spanners, we are concerned with the existence of light spanners for a given stretch. Examples of graph classes where the worst-case lightness does not depend on the number of vertices include planar graphs [2], bounded genus graphs [24], apex graphs [26], bounded pathwidth graphs [25], bounded treewidth graphs [12], and minor-free graphs [9]. Our results rely on the existence of light spanners for minor-free graphs [9] and improve on the lightness for bounded genus graphs. In particular, we study the lightness of the so-called greedy spanner [2] for graphs of bounded genus. It was shown by Filtser and

Solomon [20] that this spanner construction is existentially optimal for every class of graphs closed under taking subgraphs, which means that the optimal lightness guarantee on any such class is achieved by the greedy spanner.

Light and sparse spanners have applications in various fields. Most importantly, spanners were used to give polynomial-time approximation schemes (PTAS) for the travelling salesperson problem for various graph classes [3, 8, 9]. Note that the difference between approximations for TSP and online exploration is that, in our setting, the tour is computed on-the-fly. Indeed, in comparison to our online setting, we desire a constant approximation for an arbitrary constant, which in the TSP setting is easily obtained by traversing a minimum spanning tree twice. On the other hand, in the online setting, we are not concerned with efficiency of the algorithms which is crucial in the TSP setting. Other fields of application of spanners include distributed systems [4], routing [38], or computational biology [37].

## 2 The online graph exploration problem on minor-free graphs

In this section, we prove new upper bounds for $\textsc{Blocking}_\delta$ on $H$-minor-free graphs (Theorem 1) and for general graphs (Theorem 2). To this end, we begin by introducing the algorithm $\textsc{Blocking}_\delta$ proposed by Megow et al. [31] based on the work of Kalyanasundaram and Pruhs [29].

### 2.1 The algorithm Blocking

During the execution of an online graph exploration algorithm, a vertex is *explored* if it has been visited by the agent. A neighbor of an explored vertex is a *learned* vertex. An edge is a *boundary edge* if exactly one of its endpoints is explored. By convention, we denote boundary edges $e = (u, v)$ such that $u$ is explored and $v$ is unexplored. A path is *internally explored* if each of its internal vertices is explored. Given two learned vertices $x$ and $y$, we set the distance $d(x, y)$ to be the length of a shortest internally explored path linking $x$ with $y$. In particular, the distance may decrease during execution.

▶ **Definition 5** (Kalyanasundaram and Pruhs [29]). *Given some $\delta > 0$, we say that a boundary edge $e = (u, v)$ is* $\delta$-blocked *if there is another boundary edge $e' = (u', v')$ such that $w(e') < w(e)$ and $d(u, v') \leq (1 + \delta)w(e)$.*

The rough idea of Blocking is to perform a depth-first-traversal while ignoring all blocked edges. Whenever a previously blocked edge turns unblocked, the agent moves to and explores one such edge, and initiates a DFS-traversal from its new position. Blocking is formally specified in Algorithm 1. It is executed on an undirected, weighted, connected, and initially unexplored graph $G = (V, E, w)$ and takes as input a vertex $v$ of $G$, denoting the current position of the agent. The algorithm follows a recursive DFS-like structure and the input of the initial invocation is the start vertex.

**Algorithm 1** $\textsc{Blocking}_\delta(v)$ [29, 31].

```
1 while there is a boundary edge e = (y, x) that is not δ-blocked and such that y = v
    or e was previously blocked by some edge (u, v) do
2     traverse a shortest internally explored path from v to y
3     traverse e
4     Blocking_δ(x)
5     traverse a shortest internally explored path from x to v
```

Observe that the algorithm is correct, i.e., every vertex is explored: Assume, for the sake of contradiction, that some vertex remains unexplored when the algorithm terminates, i.e., there are still boundary edges. Let $e = (u, v)$ be a boundary edge of minimum weight. Then, $e$ is not $\delta$-blocked. Therefore, either the exploration of $u$ should have triggered the exploration of $v$, or $v$ should have been explored at the last point in time the edge turned unblocked.

## 2.2 Key properties of Blocking

Throughout the remainder of Section 2, let $G = (V, E, w)$ be a graph, $n = |V|$ be its number of vertices, $v$ the given start vertex of $G$, and $\delta = \delta(n) > 0$. We analyze the performance of $\textsc{Blocking}_\delta$ on $G$, i.e., we estimate its total cost $W_{\textsc{Blocking}}(G, v, \delta)$. For this, let $B$ be the set of boundary edges taken by $\textsc{Blocking}_\delta$, i.e., the edges traversed during the execution of line 3.

Note that the total cost of the offline optimum is bounded from below by the weight of a minimum spanning tree $w(\text{MST})$ and from above by $2w(\text{MST})$. That is, to show that $\textsc{Blocking}_\delta$ is $\rho$-competitive, it suffices to show $W_{\textsc{Blocking}}(G, v, \delta) \leq \rho \cdot w(\text{MST})$.

▶ **Observation 6** (Megow et al. [31]). *We have* $W_{\textsc{Blocking}}(G, v, \delta) \leq 2(\delta + 2)w(B)$.

**Proof.** We charge all cost incurred in lines 2,3, and 5 to the corresponding boundary edge $e \in B$. Note that the cost in line 2 is at most $(1 + \delta)w(e)$, because either we have $y = v$ such that $d_G(v, y) = 0$, or $e$ was blocked by an edge $(u, v)$, which implies $d_G(y, v) \leq (1+\delta)w(e)$. The cost in line 3 is $w(e)$ and the cost in line 5 is at most the sum of the cost in lines 2 and 3. Therefore, each edge $e$ in $B$ is charged at most $2(\delta + 2)w(e)$. ◀

In our subsequent analysis, we will frequently use a minimum spanning tree with a particular property. For this, in what follows, let $\text{MST}_B$ be a minimum spanning tree of $G$ that maximizes the number of edges in $\text{MST}_B \cap B$. As pointed out in [31], cycles in $B \cup \text{MST}_B$ are long relative to the weight of the edges they contain. Specifically, the following holds.[1]

▶ **Lemma 7.** *Let $C$ be a cycle in $B \cup \text{MST}_B$ and $e$ be an edge of $C$. Then,*

$$w(C \setminus \{e\}) > (1 + \delta)w(e).$$

**Proof.** It suffices to show the assertion for an edge of maximum weight in $C$. We first show that this edge must be in $B$, i.e., $\operatorname{argmax}\{w(e) \colon e \in C\} \subseteq B$:

Assume otherwise and let $e = (u, v) \in \operatorname{argmax}\{w(e) \colon e \in C\} \cap (\text{MST}_B \setminus B)$. Removing $e$ from $\text{MST}_B$ separates $\text{MST}_B$ into two connected components. In particular, $u$ and $v$ are in different components. Start walking in $C \setminus \{e\}$ from $u$ to $v$ and let $e'$ be the first edge that leads from $u$'s connected component in $\text{MST}_B \setminus \{e\}$ to $v$'s connected component. Then, $e' \in B \setminus \text{MST}_B$ and by maximality of $e$, we have $w(e') \leq w(e)$. Therefore, replacing $e$ by $e'$ in $\text{MST}_B$ gives another spanning tree of weight at most $w(\text{MST}_B)$. This new spanning tree has one more edge in common with $B$ than $\text{MST}_B$. This contradicts the choice of $\text{MST}_B$, so that we can assume from now on $\operatorname{argmax}\{w(e) \colon e \in C\} \subseteq B$, i.e., every edge in $\operatorname{argmax}\{w(e) \colon e \in C\}$ is *charged*, i.e., the edge is traversed in some exectution of line 3 of the algorithm.

[1] The assertion of Lemma 7 implies Claim 1 in [31], which only concerns edges not in the minimum spanning tree. However, there is a subtle flaw in the proof of Claim 1 in [31]. In fact, in that proof, it is not clear that when we replace the edge $e'$ with an edge of the fixed MST, we again obtain a minimum spanning tree. In any case, the argument above rectifies this.

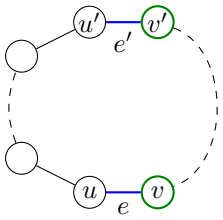



**Figure 1** Illustration of Lemma 7: The black vertices are explored and the green vertices ($v$ and $v'$) are unexplored. The blue edges ($e$ and $e'$) are boundary edges.

Let $e = (u, v)$ be the edge in $\operatorname{argmax}\{w(e) \colon e \in C\}$ that is charged last. At the time $e$ is traversed, it is a boundary edge, so that $u$ is explored but $v$ is not yet explored. Start walking in $C \setminus \{e\}$ from $u$ to $v$ and let $e' = (u', v')$ be the first edge leading from an explored vertex $u'$ to an unexplored vertex $v'$, i.e., $e'$ is another boundary edge in $C$ (cf. Figure 1).

Next, we show that $w(e') < w(e)$: Assume otherwise. By maximality of $e$, this means $w(e') = w(e)$ so that $e' \in \operatorname{argmax}\{w(e) \colon e \in C\}$. But then, we also have $e' \in B$. This contradicts the fact that $e$ is the edge in $\operatorname{argmax}\{w(e) \colon e \in C\}$ that is charged last.

Summing up, we have shown the following facts: Upon exploration of $e = (u, v)$, there is another boundary edge $e' = (u', v')$ in $C$ with $w(e') < w(e)$. Since $e$ is not blocked, this implies

$$w(C \setminus \{e\}) \geq d(u, v') > (1 + \delta)w(e). \qquad \blacktriangleleft$$

## 2.3 Connection to spanners

Next, we investigate how the performance of $\textsc{Blocking}_\delta$ is related to graph spanners. For this, note that Lemma 7 can be reformulated as follows.

▶ **Lemma 8.** *No proper subgraph of $B \cup \mathrm{MST}_B$ is a $(1 + \delta)$-spanner of $B \cup \mathrm{MST}_B$.*

The lemma relates spanners to the behavior of $\textsc{Blocking}_\delta$. However, we need to take note that the lemma applies to $B \cup \mathrm{MST}_B$ rather than the original graph $G$. A *monotone graph class* is a class of graphs $\mathcal{G}$ closed under taking subgraphs, i.e., if $G \in \mathcal{G}$ and $H$ is a subgraph of $G$, then also $H \in \mathcal{G}$. Given a graph $G$, we define $\textsc{OptSpan}_\delta(G)$ as the minimum lightness of a $(1 + \delta)$-spanner of $G$. Moreover, we set $\textsc{OptSpan}_\delta(\mathcal{G}) := \sup\{\textsc{OptSpan}_\delta(G) : G \in \mathcal{G}\}$ to be the supremum over all graphs in $\mathcal{G}$.

▶ **Theorem 9.** *For every monotone graph class $\mathcal{G}$ and every $\delta = \delta(n) > 0$, the algorithm $\textsc{Blocking}_\delta$ is $(2(\delta + 2) \cdot \textsc{OptSpan}_\delta(\mathcal{G}))$-competitive on $\mathcal{G}$.*

**Proof.** Let $G \in \mathcal{G}$. We have

$$W_{\textsc{Blocking}}(G, v, \delta) \overset{\text{Obs 6}}{\leq} 2(\delta + 2)w(B) \leq 2(\delta + 2)w(B \cup \mathrm{MST}_B). \tag{1}$$

Since $B \cup \mathrm{MST}_B$ is a subgraph of $G$, we have $B \cup \mathrm{MST}_B \in \mathcal{G}$. By Lemma 8, the only $(1 + \delta)$-spanner of $B \cup \mathrm{MST}_B$ is $B \cup \mathrm{MST}_B$ itself. Therefore,

$$w(B \cup \mathrm{MST}_B) \leq \textsc{OptSpan}_\delta(B \cup \mathrm{MST}_B) \cdot w(\mathrm{MST}_B) \leq \textsc{OptSpan}_\delta(\mathcal{G}) \cdot w(\mathrm{MST}_B). \tag{2}$$

Combined, we obtain

$$W_{\textsc{Blocking}}(G, v, \delta) \overset{(1)}{\leq} 2(\delta + 2)w(B \cup \mathrm{MST}_B) \overset{(2)}{\leq} 2(\delta + 2) \cdot \textsc{OptSpan}_\delta(\mathcal{G}) \cdot w(\mathrm{MST}_B). \qquad \blacktriangleleft$$

The theorem puts us in a position to leverage results on the lightness of spanners in order to draw conclusions regarding the competitive ratio of Blocking$_\delta$. For example, it has been shown that every planar graph contains a $(1+\delta)$-spanner of lightness at most $1+\frac{2}{\delta}$ [2]. Feeding this into Theorem 9, we conclude that Blocking$_\delta$ is $2(\delta+2)(1+2/\delta)$-competitive on planar graphs. This agrees with the bound proven in [29]. However, more generally, bounded genus graphs have light spanners. In fact, in Section 3.3, we show that every graph of genus at most $g$ contains a $(1+\delta)$-spanner of lightness at most $\left(1+\frac{2}{\delta}\right)\left(1+\frac{2g}{1+\delta}\right)$ (Theorem 4). From this, we obtain the following.

▶ **Corollary 10.** *Blocking$_\delta$ is $2(\delta+2)\left(1+\frac{2}{\delta}\right)\left(1+\frac{2g}{1+\delta}\right)$-competitive on graphs of genus at most $g$.*

Even more generally, it is known that $H$-minor-free graphs have light spanners [9]. Specifically, every $H$-minor-free graph contains a $(1+\delta)$-spanner of lightness $O\left(\frac{\sigma_H}{\delta^3}\log\left(\frac{1}{\delta}\right)\right)$ where $\sigma_H = |V(H)|\sqrt{\log|V(H)|}$. This yields a constant competitive ratio for Blocking$_\delta$ on $H$-minor-free graphs as follows.

▶ **Corollary 11.** *Blocking$_\delta$ is $2(\delta+2)\cdot O\left(\frac{\sigma_H}{\delta^3}\log\left(\frac{1}{\delta}\right)\right)$-competitive on $H$-minor-free graphs where $\sigma_H = |V(H)|\sqrt{\log|V(H)|}$.*

There are also strong bounds for general graphs. Given a graph $G$ with $n$ vertices and an integer $k \geq 1$ and $\varepsilon \in (0,1)$, $G$ contains a $(2k-1)(1+\varepsilon)$-spanner of lightness $O_\varepsilon\left(n^{1/k}\right)$ [11], where the notation $O_\varepsilon$ indicates that the constant factor hidden in the $O$-notation depends on $\varepsilon$. This gives us the following.

▶ **Corollary 12.** *Given an integer $k = k(n) \geq 1$ and $\varepsilon \in (0,1)$, Blocking$_{(2k-1)(1+\varepsilon)}$ is $2((2k-1)(1+\varepsilon)+2)\cdot O_\varepsilon\left(n^{1/k}\right)$-competitive on every graph.*

In particular, by suitably choosing $\delta$, we obtain the following.[2]

▶ **Corollary 13.**

**a)** *Blocking$_2$ is $16(1+\frac{2}{3}g)$-competitive on graphs of genus at most $g$.*

**b)** *For every constant $\delta > 0$ and every graph $H$, Blocking$_\delta$ is constant-competitive on $H$-minor-free graphs.*

**c)** *Blocking$_{\log(n)}$ is $O(\log(n))$-competitive on every graph.*

For the case of planar graphs, part a) matches the best known bounds on planar graphs [29, 31]. For general surfaces, it slightly improves on the best known bound of $16(1+2g)$ on bounded genus graphs [31]. Part b) is the first constant bound on minor-free graphs, and part c) is the first $O(\log(n))$ bound for Blocking.

## 2.4 Lower bounds for Blocking

Next, we investigate lower bounds for Blocking when $\delta$ is allowed to depend on the input size. In [31], it was shown that the competitive ratio of Blocking$_\delta$ on general graphs is at least $\Omega(n^{1/(\delta+4)})$ when $\delta$ is a constant. We begin by observing that this can be generalized to non-constant $\delta$ that are not too large.

▶ **Observation 14.** *Suppose $\delta = \delta(n) > 0$ such that $\delta^{2\delta+8} = o(n)$. Then, the competitive ratio of Blocking$_\delta$ is at least $\Omega(\delta \cdot n^{1/(\delta+4)})$.*

[2] All missing proofs are deferred to the full version.

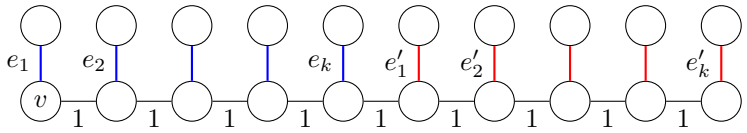



**Figure 2** Illustration of the lower bound construction for BLOCKING$_\delta$ (Lemma 15). The light edges (depicted in blue) are of weight 1 and the heavy edges (depicted in red) are of weight $\frac{k+1}{\delta+1}$.

Note that $2\delta + 8 \leq \log(n)/\log(\log(n))$ implies that

$$\delta^{2\delta+8} \leq \left(\frac{\log(n)}{\log(\log(n))}\right)^{\frac{\log(n)}{\log(\log(n))}} = \left(\frac{1}{\log(\log(n))}\right)^{\frac{\log(n)}{\log(\log(n))}} \cdot e^{\log(\log(n))\frac{\log(n)}{\log(\log(n))}}$$
$$= \left(\frac{1}{\log(\log(n))}\right)^{\frac{\log(n)}{\log(\log(n))}} \cdot n = o(n),$$

i.e., the prerequisites of Observation 14 are fulfilled. Moreover, $\Omega(\delta n^{1/(\delta+4)}) \geq \Omega(\log(n))$ for every $\delta = \delta(n)$. Therefore, Observation 14 shows that BLOCKING$_\delta$ has competitive ratio in $\Omega(\log(n))$ whenever $\delta = o(\log(n)/\log\log(n))$. In particular, this shows the first part of Theorem 3b.

Next, we give another lower bound which shows that the parameter $\delta$ cannot be chosen too large either (the second part of Theorem 3b).

▶ **Lemma 15.** *Suppose $\delta = \delta(n) \in (0, \frac{n-4}{4})$. The competitive ratio of* BLOCKING$_\delta$ *is at least $\Omega(\delta)$, even on trees.*

**Proof sketch.** It is not difficult to check that, on the graph illustrated in Figure 2, the cost of BLOCKING is asymptotically $\delta$ times the cost of the offline optimum. A complete proof can be found in the full version. ◀

To conclude our lower bound arguments for BLOCKING$_\delta$, observe that, for $\delta \geq \frac{n-4}{4}$, the behavior of BLOCKING$_\delta$ closely resembles the behavior of the algorithm hDFS [31]. In fact, it is not difficult to check that, on the lower bound construction for hDFS given in [31, Theorem 5], after proceeding to edges of weight more than 16, BLOCKING$_\delta$ takes the exact same route as hDFS, if $\delta \geq \frac{n-4}{4}$. Therefore, we obtain the following.

▶ **Observation 16.** *For $\delta \geq \frac{n-4}{4}$, the competitive ratio of* BLOCKING$_\delta$ *is at least $\Omega(\log(n))$.*

We can now combine the lower bound constructions from this section to prove Theorem 3.

**Proof of Theorem 3.** We begin with proving part b). In Observation 14, we have seen that the competitive ratio of BLOCKING$_\delta$ is at least $\Omega(\log(n))$ if $\delta \in o(\log(n)/\log\log(n))$. By Lemma 15, we obtain the same lower bound for every $\delta$ in the range from $\Omega(\log(n))$ to $(n-4)/4$, and by Observation 16, we obtain the lower bound for $\delta$ at least $(n-4)/4$. Therefore, this proves the assertion of Theorem 3b. For part a), note that part b) implies that a competitive ratio of $o(\log(n))$ is only possible for $\delta$ in the range from $\Omega(\log(n)/\log\log(n))$ to $o(\log(n))$. Using Observation 14 in this range implies the assertion of Theorem 3a. ◀

## 3 Graph spanners in bounded genus graphs

In this section, we prove Theorem 4 about the existence of light spanners in bounded genus graphs. For this, we begin by introducing the greedy spanner.

## 3.1 The greedy spanner

The *greedy* $(1+\varepsilon)$*-spanner* was suggested by Althöfer et al. [2] and is formally defined as the output of Algorithm 2. After ordering the edges by weight, it iteratively adds edges if they are short in comparison to the distance of their endpoints in the graph constructed so far.

**Algorithm 2** GreedySpanner$(G=(V,E,w),\varepsilon)$.

```
sort E = {e_1, ..., e_m} such that w(e_1) ≤ w(e_2) ≤ ··· ≤ w(e_m)
H ← (V, ∅)
for i ← 1, ..., m do
    if d_H(u_i, v_i) > (1+ε)w(e_i), where e_i = (u_i, v_i) then
        H ← H ∪ {e_i}
return H
```

Note that the resulting graph $H$ is indeed a $(1+\varepsilon)$-spanner of $G$. The output of the algorithm actually depends on the chosen order of the edges. In particular, when edge weights appear multiple times, there may be several possible outputs. However, this will not be important in our context. When we refer to *the* greedy spanner, we mean that we arbitrarily fix some output of the algorithm.

The greedy spanner fulfills the following two key properties: First, the algorithm implicitly executes Kruskal's algorithm for finding a minimum spanning tree, i.e., it adds all edges to $H$ that Kruskal's algorithm adds. With this, we obtain the following.

▶ **Observation 17.** *The greedy spanner contains all edges of some minimum spanning tree of the input graph.*

The second key property, in fact, resembles the property of Blocking$_\delta$ in Lemma 7.

▶ **Observation 18** (Althöfer et al. [2])**.** *For every cycle $C$ in the greedy spanner $H$ and every edge $e$ of $C$, we have $w(C\setminus\{e\})>(1+\varepsilon)w(e)$. In other words, no proper subgraph of $H$ is a $(1+\varepsilon)$-spanner of $H$.*

**Proof.** Let $C$ be a cycle in the greedy spanner. Let $e=(u,v)$ be the edge in $C$ that is added last. At the time it is added, we have $(1+\varepsilon)w(e)<d_H(u,v)\le w(C\setminus\{e\})$ by definition of the algorithm. Since all other edges in $C$ have lower or equal weight than $e$, the property is fulfilled for them as well. ◀

## 3.2 Spanners in planar graphs

Before investigating spanners in bounded genus graphs, we illustrate the technique for the special case of planar graphs, giving an alternate proof of the following result.

▶ **Theorem 19** (Althöfer et al. [2])**.** *For every planar graph $G$ and $\varepsilon>0$, the greedy $(1+\varepsilon)$-spanner of $G$ has lightness at most $1+\frac{2}{\varepsilon}$.*

Our proof uses similar ideas as in [31, Theorem 1] and is based on the following main idea: Fix an embedding of the greedy spanner in the plane and, in a suitable way, partition the greedy spanner into facial cycles, i.e., cycles that form the boundary of a face. Then use the fact that none of these cycles are short (cf. Observation 18).

▶ **Lemma 20.** *Let $G$ be a planar graph, $H$ be the greedy $(1+\varepsilon)$-spanner of $G$ and* MST *be a minimum spanning tree of $H$. Fix an embedding of $H$ in the plane. Then, we can associate with every edge $e \in H \setminus \text{MST}$ a facial cycle $C_e$ containing $e$, so that $C_e \neq C_{e'}$ for $e \neq e'$.*

Next, we illustrate how this can be combined with the fact that the greedy spanner does not contain short cycles (cf. Observation 18).

▶ **Lemma 21.** *Let $G$ be a graph and $H$ be the greedy $(1+\varepsilon)$-spanner of $G$. Let $D$ be a subgraph of $G$ such that we can associate with every edge $e \in H \setminus D$ a cycle $C_e$ of $H$ containing $e$, with the property that $\sum_{e \in H \setminus D} w(C_e) \leq 2w(H)$. Then, $w(H) \leq \left(1 + \frac{2}{\varepsilon}\right) w(D)$.*

Next, we show how this implies Theorem 19.

**Proof of Theorem 19.** Let $G$ be a planar graph, let $H$ be the greedy $(1+\varepsilon)$-spanner of $G$, and let MST denote a minimum spanning tree of $H$. By Observation 17, MST is also a minimum spanning tree of $G$, so that it suffices to show $w(H) \leq \left(1 + \frac{2}{\varepsilon}\right) w(\text{MST})$. Since $G$ is planar, its subgraph $H$ is planar as well. Let us fix an embedding of $H$ on the plane such that no two edges cross. By Lemma 20, there is a facial cycle $C_e$ for every edge $e \in H \setminus \text{MST}$ such that $C_e \neq C_{e'}$ for $e \neq e'$. As every edge of $H$ is contained in at most two facial cycles, we have $\sum_{e \in H \setminus \text{MST}} w(C_e) \leq 2w(H)$. Therefore, we can apply Lemma 21 with $D = \text{MST}$ and obtain $w(H) \leq \left(1 + \frac{2}{\varepsilon}\right) w(\text{MST})$. ◀

## 3.3 Generalization to bounded genus graphs

The *genus* of a graph $G$ is the smallest integer $g$ such that $G$ can be embedded on an orientable surface of genus $g$. In this subsection, we study light spanners for the class of bounded genus graphs and prove Theorem 4. We begin by recalling the theorem.

▶ **Theorem 4** (restated). *For every $\varepsilon > 0$, the greedy $(1+\varepsilon)$-spanner of a graph of genus $g$ has lightness at most $\left(1 + \frac{2}{\varepsilon}\right)\left(1 + \frac{2g}{1+\varepsilon}\right)$.*

Our proof is based on similar arguments as in [31, Theorem 2] and the main idea is roughly as follows: Given an embedding of the greedy spanner on a surface of genus $g$, first cut the surface along several edges such that we obtain a disk. Then, we can proceed along similar lines as for Theorem 19. In this work, we estimate more carefully the weight of the edges along which we cut so that we obtain a slightly improved bound than in [31]. We will use the following topological lemma for the first step.

▶ **Lemma 22.** *Let $G$ be an unweighted connected graph of genus (exactly) $g \geq 1$. Fix an embedding of $G$ on an orientable surface of genus $g$ and let $T$ be a spanning tree of $G$. Then, there is a subgraph $D$ of $G$ with $T \subseteq D$ and $|E(D) \setminus E(T)| \leq 2g$ such that, in the inherited embedding of $D$, there is only a single face and the edges in $D$ bound a topological disk.*[3]

**Proof.** It is a standard fact from topology that, on a surface of genus $g$, one can embed precisely $2g$ closed curves that are non-separating, i.e., it is possible to draw $2g$ cycles on the surface such that cutting along all of them does not disconnect the surface. Every collection of $2g$ curves that are non-separating bounds a topological disk (see Figure 3).[4]

[3] A topological disk is a surface homeomorphic to a 2-dimensional disk. Intuitively, a topological disk is a continuous deformation of a 2-dimensional disk.

[4] This can be proven as follows: The Euler characteristic of a surface of genus $g$ is $2 - 2g$ [27, Section 2.2] and cutting along a non-separating closed curve increases the Euler characteristic by 1.

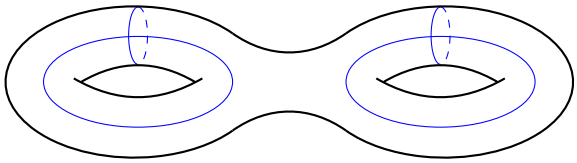


**Figure 3** surface of genus 2 with 4 non-separating cycles bounding a topological disk.

We construct the set $D$ greedily as follows (see Figure 4): Initially, let $D := T$. Ignoring all edges in $G \setminus D$, we have only a single face. Note that every edge in $G \setminus D$ closes a cycle with $D$. If we find an edge which only closes non-separating cycles, i.e, does not separate the surface into two faces, we add it to $D$. After this, the edges of $D$ still only bound a single face. We repeat this step until we cannot find further edges whose addition would separate the surface into multiple faces.

Since there are at most $2g$ cycles on a surface of genus $g$ that are non-separating, we have $|E(D) \setminus E(T)| \leq 2g$. It is left to show that $D$ bounds a disk. By maximality of $D$, every edge $e \in G \setminus D$ is separating when added to $D$, i.e., in the inherited embedding of $D \cup \{e\}$, the edge $e$ is incident to two faces. In particular, $e$ is incident to two faces in the inherited embedding of every supergraph of $D$.

Consider again the embedding of the entire graph $G$. It is known from topological graph theory that a minimal genus embedding of a connected graph is cellular, i.e., every face of the embedding of $G$ is a topological disk [39] (see [34, Proposition 3.4.1]). Since every edge $e \in G \setminus D$ is incident to two distinct faces, its removal merges the two corresponding disks along a connected part of their common boundary, which yields another disk. Iteratively removing all edges in $G \setminus D$ in this way, we thus obtain a cellular embedding of $D$. Since, by construction, $D$ induces only a single face, we obtain that $D$ bounds a topological disk.

For an illustration of the construction, consider Figure 4. In the example in the left column, the two green edges enclose non-separating cycles, whereas all blue edges close separating cycles. In the example in the right column, the half-dotted green edge in $D$ could be replaced by the blue edge between $u$ and $v$. ◀

Now, we have all the prerequisites in place to prove Theorem 4. The main idea is to give a similar construction as in Lemma 20 to partition the greedy spanner into facial cycles. Before delving into the proof, let us briefly comment on why Lemma 22 is not a reduction to the planar case, i.e., we cannot use the same construction as in Lemma 20.

Recall that the key ingredient of Lemma 20 was to define a partial order in which an edge $e'$ precedes another edge $e$ if $e'$ is embedded on the inside of the cycle that $e$ closes with MST. In the bounded genus case, if the cycle closed by $e$ is non-separating, there is no such thing as "the inside" of the cycle. For example, consider the edge $(u, v)$ in the right column of Figure 4 and the cycle it closes with MST. This cycle does not have an "inside" and cannot be decomposed into multiple faces. In particular, the cycle disappears after cutting the surface along $D$. However, it separates the disk bounded by $D$ into two parts. Therefore, we have to consider cycles that include edges of $D \setminus \text{MST}$.

**Proof of Theorem 4.** Let $G$ be some graph of genus $g$. Let $H$ be the greedy $(1 + \varepsilon)$-spanner of $G$ and let MST denote a minimum spanning tree of $H$. By Observation 17, we know that MST is also a minimum spanning tree of $G$, so that it suffices to show $w(H) \leq \left(1 + \frac{2}{\varepsilon}\right)\left(1 + \frac{2g}{1+\varepsilon}\right) w(\text{MST})$.

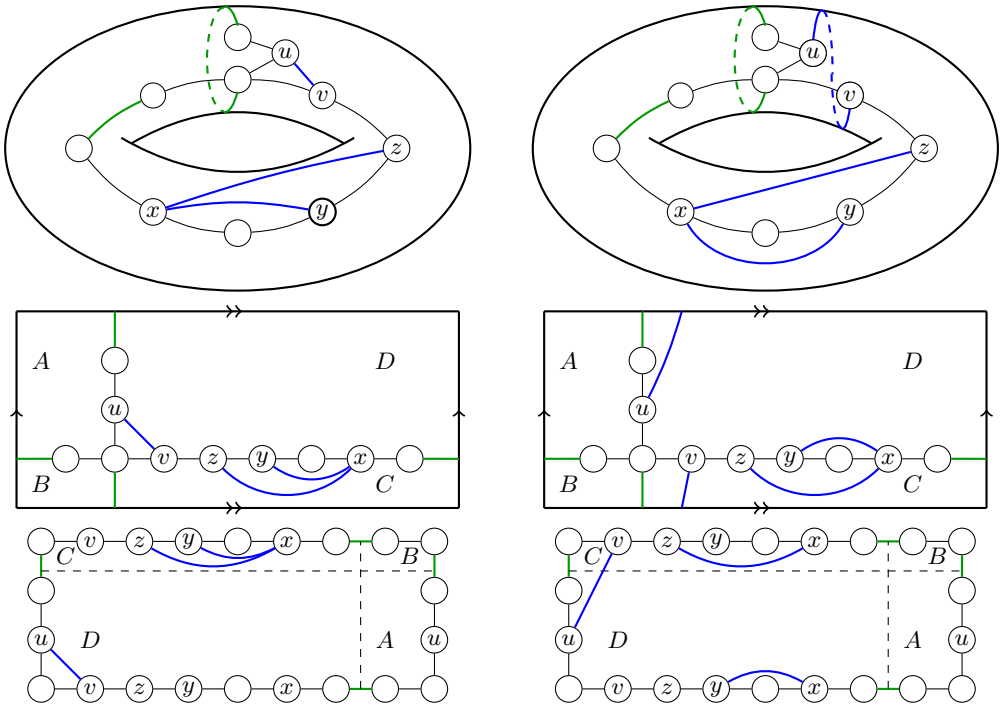



**Figure 4** The two columns show the construction of $D$ for the same graph with two different embeddings. The black edges belong to $T$, the green edges to $D \setminus T$, and the blue edges to $G \setminus D$. In each column, the first subfigure shows the embedding on the torus. The second subfigure shows a different representation: The torus is obtained by gluing together the opposite sides of the rectangle. The last subfigure shows the disk obtained by cutting the surface along $D$. Note that it contains every edge of $D$ twice and therefore, every vertex up to 4 times. However, note that the embedding specifies between which copies of the vertices the blue edges have to be drawn. The capital letters $A, B, C, D$ denote areas of the torus and are included for better orientation: Leaving area $A$ to the left leads to area $D$, leaving $A$ to the top leads to $B$ and so on.

Let $g'$ be the genus of $H$. If $g' = 0$, the assertion follows directly by Theorem 19. Therefore, we assume from now on $g' \geq 1$. Note that $g' \leq g$ because $H$ is a subgraph of $G$. Fix an embedding of $H$ on an orientable closed surface of genus $g'$ such that no two edges cross. By Lemma 22, there is a subgraph $D$ of $H$ with MST $\subseteq D$ such that

$$|E(D) \setminus E(\text{MST})| \leq 2g' \leq 2g \tag{3}$$

and such that the edges of $D$ induce only one face and bound a topological disk. Next, observe that, for every edge $e$ in $H \setminus \text{MST}$, we have $w(e) \leq w(\text{MST})/(1+\varepsilon)$: Every edge $e$ in $H \setminus \text{MST}$ closes a cycle $C$ together with the edges of MST. Using Observation 18, we obtain

$$w(e) < \frac{w(C \setminus \{e\})}{1+\varepsilon} \leq \frac{w(\text{MST})}{1+\varepsilon}.$$

In particular, this is fulfilled for edges in $D \setminus \text{MST}$. Combining this with (3), we obtain

$$w(D) \leq \left(1 + \frac{2g}{1+\varepsilon}\right) w(\text{MST}). \tag{4}$$

The next step is to bound the weight of $H$ by $(1 + 2/\varepsilon)w(D)$. For this, we use a similar construction as in Lemma 20 and show that it is possible to iteratively choose an edge $e$ in $H \setminus D$ which, together with the edges of $D$ and the edges chosen in previous iterations, closes a facial cycle $C_e$ in the embedding of $H$.

In each iteration, we find a suitable edge as follows: Pick an arbitrary edge $e$ of $H \setminus D$. If it defines a facial cycle together with $D$ and edges chosen in previous iterations, we can simply choose $e$. Assume this is not the case. Note that $e$ cuts the disk bounded by $D$ in two parts and both contain edges in $H \setminus D$ to which no cycles have been assigned yet (otherwise $e$ would close a suitable facial cycle). Pick the part whose boundary with $D$ contains fewer edges (breaking ties arbitrarily) and pick a new edge $e'$ in $H \setminus D$ which lies inside this half and has not yet been chosen in a previous iteration. Note that $e'$ again cuts the disk in two parts and the boundary of the smaller part contains fewer edges of $D$ than in the step before. Therefore, by repeating the steps above, we will end up with a suitable edge after finitely many steps. For example, on the left side of Figure 4, if we pick $e = (x, z)$, we will set $e' = (x, y)$ and this edge is suitable. After this, we can assign a facial cycle to $(x, z)$ and then to $(u, v)$. In the instance on the right, we can assign the cycles to the blue edges in any order.

Note that, in this construction, no two edges are assigned the same facial cycle. As every edge is contained in at most two facial cycles, we have

$$\sum_{e \in H \setminus D} w(C_e) \leq 2w(H). \tag{5}$$

Therefore, we can now apply Lemma 21 and obtain

$$w(H) \overset{\text{Lem 21}}{\leq} \left(1 + \frac{2}{\varepsilon}\right) w(D) \overset{(4)}{\leq} \left(1 + \frac{2}{\varepsilon}\right)\left(1 + \frac{2g}{1+\varepsilon}\right) w(\text{MST}). \qquad \blacktriangleleft$$

Recall that Grigni showed that every graph of genus $g \geq 1$ contains a $(1+\varepsilon)$-spanner of lightness at most $1 + (12g - 4)/\varepsilon$ [24]. Let us briefly comment on how our bound compares to Grigni's bound. For this, note that, for $g \geq 1$,

$$\left(1 + \frac{2}{\varepsilon}\right)\left(1 + \frac{2g}{1+\varepsilon}\right) = 1 + \frac{2}{\varepsilon} + \frac{2g}{1+\varepsilon} + \frac{4g}{\varepsilon(1+\varepsilon)} < 1 + \frac{2g}{\varepsilon} + \frac{2g}{\varepsilon} + \frac{4g}{\varepsilon} = 1 + \frac{8g}{\varepsilon}.$$

Therefore, our bound is stronger than Grigni's bound for every $g \geq 1$. Moreover, in the planar case (i.e., $g = 0$), we obtain a lightness of $1 + \frac{2}{\varepsilon}$. It was shown by Althöfer et al. [2, Theorem 5] that this is best possible, i.e., our bound is tight for planar graphs. Note that the worst-case lightness for spanners of graphs of genus $g$ has to increase in $g$, since not every graph admits a light spanner. For example, for every $k \geq 3$ and almost all $n$, there is a graph on $n$ vertices with girth at least $k$ and at least $\frac{1}{4}n^{1+\frac{1}{k}}$ edges [32, Theorem 6.6].

## 4 Open problems

The key question in online graph exploration is whether the problem admits a constant-competitive algorithm [29]. While this problem remains open, our results suggest steps that might be needed towards a resolution of this question. Firstly, we have shown that the online graph exploration problem allows for a constant-competitive algorithm on graphs admitting a light spanner, in particular, minor-free graphs. This suggests that, for proving a non-constant general lower bound on the competitive ratio, one might require dense high-girth graphs or expanders [30]. Not even a competitive ratio of $o(\log(n))$ has yet been attained, and our results eliminate $\textsc{Blocking}_\delta$, for most values of $\delta$, as a candidate for achieving this. It remains to close the gap between $\delta \in o(\log(n)/\log\log(n))$ and $\delta \in \Omega(\log(n))$.

Regarding spanners, we gave an improved upper bound on the lightness of spanners in bounded genus graphs. It is a natural question whether our bound is already tight for $g \geq 1$ or can further be improved. In particular, it is unclear whether the worst-case lightness for a fixed stretch must depend linearly on $g$.

## References


1 Susanne Albers and Monika R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

2 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

3 Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 33–41, 1998.

4 Baruch Awerbuch, Alan Baratz, and David Peleg. Cost-sensitive analysis of communication protocols. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 177–187, 1990.

5 Alexander Birx, Yann Disser, Alexander V. Hopp, and Christina Karousatou. An improved lower bound for competitive graph exploration. *Theoretical Computer Science*, 868:65–86, 2021.

6 Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online TSP on the line. *ACM Transactions on Algorithms*, 17(1):3:1–3:58, 2021.

7 Vincenzo Bonifaci and Leen Stougie. Online k-server routing problems. *Theory of Computing Systems*, 45(3):470–485, 2009.

8 Glencora Borradaile, Erik D. Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.

9 Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. Minor-free graphs have light spanners. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 767–778, 2017.

10 Sebastian Brandt, Klaus-Tycho Foerster, Jonathan Maurer, and Roger Wattenhofer. Online graph exploration on a restricted graph class: Optimal solutions for tadpole graphs. *Theoretical Computer Science*, 839:176–185, 2020.

11 Shiri Chechik and Christian Wulff-Nilsen. Near-optimal light spanners. *ACM Transactions on Algorithms*, 14(3):1–15, 2018.

12 Erik D. Demaine, MohammadTaghi HajiAghayi, and Bojan Mohar. Approximation algorithms via contraction decomposition. *Combinatorica*, 30(5):533–552, 2010.

13 Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. *Journal of Graph Theory*, 32(3):265–297, 1999.

14 Dariusz Dereniowski, Yann Disser, Adrian Kosowski, Dominik Paj, and Przemysław Uznański. Fast collaborative graph exploration. *Information and Computation*, 243:37–49, 2015.

15 Yann Disser, Jan Hackfeld, and Max Klimm. Tight bounds for undirected graph exploration with pebbles and multiple agents. *Journal of the ACM*, 66(6):40(41), 2019.

16 Yann Disser, Frank Mousset, Andreas Noever, Nemanja Skoric, and Angelika Steger. A general lower bound for collaborative tree exploration. *Theoretical Computer Science*, 811:70–78, 2020.

17 Stefan Dobrev, Rastislav Královič, and Euripides Markou. Online graph exploration with advice. In *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 267–278, 2012.

18 Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. Robustification of online graph exploration methods. In *Proceedings of the 36th Conference on Artificial Intelligence (AAAI)*, pages 9732–9740, 2022.

19 Paul Erdős. Extremal problems in graph theory. In *Proceedings of the Symposium on Theory of Graphs and its Applications*, pages 29–36, 1963.

20 Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. *SIAM Journal on Computing*, 49(2):429–447, 2020.

21 Rudolf Fleischer and Gerhard Trippen. Exploring an unknown graph efficiently. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, pages 11–22, 2005.

22 Klaus-Tycho Foerster and Roger Wattenhofer. Lower and upper competitive bounds for online directed graph exploration. *Theoretical Computer Science*, 655:15–29, 2016.

23 Robin Fritsch. Online graph exploration on trees, unicyclic graphs and cactus graphs. *Information Processing Letters*, 168:106096, 2021.

24 Michelangelo Grigni. Approximate TSP in graphs with forbidden minors. In *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1853, pages 869–877, 2000.

25 Michelangelo Grigni and Hao-Hsiang Hung. Light spanners in bounded pathwidth graphs. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 467–477, 2012.

26 Michelangelo Grigni and Papa Sissokho. Light spanners and approximate TSP in weighted graphs with forbidden minors. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 852–857, 2002.

27 Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2000. URL: `https://cds.cern.ch/record/478079`.

28 Cor A.J. Hurkens and Gerhard J. Woeginger. On the nearest neighbor rule for the traveling salesman problem. *Operations Research Letters*, 32(1):1–4, 2004.

29 Bala Kalyanasundaram and Kirk R. Pruhs. Constructing competitive tours from local information. *Theoretical Computer Science*, 130(1):125–138, 1994.

30 Michael Krivelevich and Benjamin Sudakov. Minors in expanding graphs. *Geometric and Functional Analysis*, 19(1):294–331, 2009.

31 Nicole Megow, Kurt Mehlhorn, and Pascal Schweitzer. Online graph exploration: New results on old and new algorithms. *Theoretical Computer Science*, 463:62–72, 2012.

32 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2017.

33 Shuichi Miyazaki, Naoyuki Morimoto, and Yasuo Okabe. The online graph exploration problem on restricted graphs. *IEICE transactions on information and systems*, 92(9):1620–1627, 2009.

34 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

35 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.

36 Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

37 Daniel Russel and Leonidas J. Guibas. Exploring protein folding trajectories using geometric spanners. In *Pacific Symposium on Biocomputing (PSB)*, pages 42–53. World Scientific, 2005.

38 Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang. Light graphs with small routing cost. *Networks*, 39(3):130–138, 2002.

39 John William Theodore Youngs. Minimal imbeddings and the genus of a graph. *Journal of Mathematics and Mechanics*, pages 303–315, 1963.

# Learning-Augmented Online TSP on Rings, Trees, Flowers and (Almost) Everywhere Else

**Evripidis Bampis**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Bruno Escoffier**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
Institut Universitaire de France, Paris, France

**Themis Gouleakis**
National University of Singapore, Singapore

**Niklas Hahn**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Kostas Lakis**
ETH Zürich, Switzerland

**Golnoosh Shahkarami**
Max-Planck-Institut für Informatik, Universität des Saarlandes, Saarbrücken, Germany

**Michalis Xefteris**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

## Abstract

We study the Online Traveling Salesperson Problem (OLTSP) with predictions. In OLTSP, a sequence of initially unknown requests arrive over time at points (locations) of a metric space. The goal is, starting from a particular point of the metric space (the origin), to serve all these requests while minimizing the total time spent. The server moves with unit speed or is "waiting" (zero speed) at some location. We consider two variants: in the open variant, the goal is achieved when the last request is served. In the closed one, the server additionally has to return to the origin. We adopt a prediction model, introduced for OLTSP on the line [24], in which the predictions correspond to the locations of the requests and extend it to more general metric spaces.

We first propose an oracle-based algorithmic framework, inspired by previous work [14]. This framework allows us to design online algorithms for general metric spaces that provide competitive ratio guarantees which, given perfect predictions, beat the best possible classical guarantee (*consistency*). Moreover, they degrade gracefully along with the increase in error (*smoothness*), but always within a constant factor of the best known competitive ratio in the classical case (*robustness*).

Having reduced the problem to designing suitable efficient oracles, we describe how to achieve this for general metric spaces as well as specific metric spaces (rings, trees and flowers), the resulting algorithms being tractable in the latter case. The consistency guarantees of our algorithms are tight in almost all cases, and their smoothness guarantees only suffer a linear dependency on the error, which we show is necessary. Finally, we provide robustness guarantees improving previous results.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms; Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** TSP, Online algorithms, Learning-augmented algorithms, Algorithms with predictions, Competitive analysis

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.12

**Related Version** *Full Version*: `https://arxiv.org/pdf/2305.02169.pdf` [13]



31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 12; pp. 12:1–12:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Acknowledgements** This work was partially funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR). Kostas Lakis gratefully acknowledges financial support from the Foundation for Education and European Culture, Athens, Greece. Themis Gouleakis was supported by the National Research Foundation Fellowship for AI (Award NRF-NRFFAI-0002), an Amazon Research Award, and a Google South & Southeast Asia Research Award.

## 1 Introduction

In the classical Traveling Salesperson Problem (TSP), we are given a set of locations as well as the pairwise distances between them and the objective is to find a shortest tour visiting all the locations. TSP is one of the most fundamental and well studied problems in Computer Science [31]. We focus on the online version of the problem in a metric space, the Online Traveling Salesperson Problem (OLTSP), introduced in the seminal paper of Ausiello et al. [10]. In OLTSP, the input arrives over time, i.e., new requests (locations) that have to be visited by the traveler (or server) will appear during the travel. The time in which a request is communicated to the traveler is called its release time (or release date). The objective is the minimization of the total traveled time assuming that at any time the traveler either moves at unit speed or is "waiting" (zero speed) at some location[1]. We study both the *closed* variant, where the server is required to return to the origin after serving all requests, and the *open* variant, where the server does not have to return to the origin after serving all the requests. A series of papers considered many variations of OLTSP in different metric spaces (general metric space [10, 14], the line [10, 18, 24], the semi-line [9, 14, 19], the ring [14, 28] and the star [14]).

The motivation of studying OLTSP and its variations comes from applications in many different domains, such as e.g. logistics and robotics [8, 38]. In the framework of competitive analysis, the performance of an online algorithm is usually evaluated using the competitive ratio which is defined as the maximum ratio between the cost of the online algorithm and the cost of an optimal offline algorithm, which by definition has knowledge of the entire input in advance, over all input instances. However, it is admitted that the competitive analysis approach can be overly pessimistic as it is calculated considering worst-case instances, giving a lot of power to the adversary. Hence, many papers try to limit the power of the adversary [19], or give extra knowledge and hence more power to the online algorithm [1, 14, 27].

More recently, the framework of *Learning-Augmented (LA) algorithms* has emerged due to the vibrant successes of Machine Learning methods and Artificial Intelligence in predicting and learning the unknown (i.e., future inputs in the case of online algorithms) based on data [33]. In this line of research, the goal is to utilize predictions of the future input that have potentially been acquired using a learning algorithm, in order to have provably improved competitive ratio in the case that the predictions are accurate enough, while maintaining worst-case guarantees even if the prediction error is arbitrarily large. In particular, a (possibly erroneous) prediction of the input is given to the algorithm and the goal is to design algorithms with a good performance guarantee when the prediction is accurate (consistency), a not too bad (and bounded) performance when the prediction is wrong (robustness) and a gradual deterioration of the competitive ratio with respect to the prediction error (smoothness). We give more precise definitions in Section 2.

[1] Note that this choice of possible speeds is w.l.o.g., as any setting where the maximum speed is bounded can be reduced to this setting. Specifically, if the maximum speed is some $S > 0$, we can normalize the maximum speed to be 1 and multiply all distances by $S$. Also, our setting can simulate any other setting where the algorithm is free to chose either the unit speed of some other speed $0 < s < 1$. Moving at speed $s$ is equivalent to dividing the time into intervals of length $dt \to 0$ and moving at unit speed only for time $s \cdot dt$ within each interval.

Recent works have proposed a variety of approaches to tackle OLTSP with predictions (LA-OLTSP). In [24], Gouleakis et al. studied a learning-augmented framework for OLTSP on the line. They introduced a prediction model in which the predictions correspond to the locations of the requests. They proposed LA algorithms for both the closed and the open variants that are consistent, smooth and robust. Bampis et al., in [14], considered the case of perfect predictions of the locations and studied different metric spaces (general metric space, semi-line, ring, and star) and proposed competitive online algorithms and lower bounds. In [26], Hu et al. proposed three different prediction models for OLTSP. In two of their models, each request is associated to a prediction for both its release time and its location while in the third one the prediction is just the release time of the last request. In [17], Bernardini et al. studied OLTSP with predictions of both the release time and the location of each request. They introduced a new error measure, the cover error, and they also considered other online graph problems. More recently, Chawla and Christou [20] studied the Online Time-Windows TSP with predictions of release time and location of the requests.

In this work, we adopt the prediction model of [24]. We propose a general oracle-based framework that allows us to design consistent, smooth and robust LA algorithms for both the closed and open variants of the problem for different metric spaces. We also provide some interesting lower bounds.

## 1.1 Our contributions and techniques

In this paper, we propose a novel approach to improve the competitive ratio of OLTSP using predictions concerning the locations of requests in general metric spaces. Our algorithms provide tight competitive ratio guarantees in most cases. Moreover, we show how to get polynomial-time/FPT algorithms in specific metrics, namely rings, trees and flowers.

Bampis et al. [14] gave an algorithm for general metrics with a competitive ratio of 3/2 for the case of perfect predictions (known locations), which is tight. First, in Section 3, we modify this algorithm (still under the assumption of perfect predictions) and introduce our main oracle-based 3/2-competitive framework, which we call *Strategically Wait And Go* (`SWAG`, for pseudocode see Algorithm 1). The main idea is to consider a suitable subset of permutations of the requests, referred to as *Dominating permutations*, given by a so-called *Domination oracle* instead of all the permutations. This allows for a reduction of the running time, since the bottleneck is located in the cardinality of the set of considered permutations. This restriction of the permutation set preserves the consistency of 3/2.

Then, we introduce our main algorithm, *Learning-Augmented Strategically Wait And Go* (`LA-SWAG`, for pseudocode see Algorithm 2), which does not assume perfect predictions. `LA-SWAG` is consistent, smooth, and robust. More formally, in Section 4 we show the following.

▶ **Theorem 1** (Consistency and Smoothness). *`LA-SWAG` has a competitive ratio of at most* $3/2 + 5\eta$ *for both closed and open LA-OLTSP.*

Here, $\eta$ is the error of the prediction (defined formally later) that captures the normalized sum of distances between predicted and actual locations of the requests. Note that for $\eta = 0$, we get a consistency of 3/2, which is tight for all cases except the open variant on trees.

Additionally, we show a smoothness lower bound of $3/2 + \eta/2$ for the open variant with $\eta \in [0, 1/3]$ (Proposition 14), implying that a linear dependency on $\eta$ is required.

Regarding robustness, the algorithm in [24] achieves 3-robustness on the line. `LA-SWAG` improves this bound for general metrics and further so in specific metrics.

▶ **Theorem 2** (Robustness-Closed). *`LA-SWAG` is* 2.75*-robust for closed LA-OLTSP in general metric spaces, and* 2.5*-robust in Euclidean spaces and in trees.*

▶ **Theorem 3** (Robustness-Open). *LA-SWAG is* $(3 - 1/6)$*-robust for open LA-OLTSP in general metric spaces, and* $(3 - 1/3)$*-robust in trees.*

Our analysis for 2.5-robustness and $(3 - 1/3)$-robustness is tight even on the line. Moreover, we show a negative result concerning the consistency/robustness trade-off of any algorithm for the open variant (Lemma 15).

The main technical contribution of our work is the implementation of the domination oracles we have referred to. On a high level, we say that a permutation $\pi_{dom}$ dominates another permutation $\pi$ at time $t$, if the following conditions hold. Assuming $q$ is the first unreleased request in $\pi$ at time $t$, the distance traveled up to $q$ is not longer in $\pi_{dom}$, and also a superset of the requests preceding $q$ in $\pi$ is visited before $q$. Moreover, $\pi_{dom}$ induces a not longer path than $\pi$. These two key facts allow us to preserve 3/2-consistency.

For general metrics, we achieve this domination using a very similar idea as the one employed in the definition of the $O(n^2 2^n)$ dynamic programming solution of the classical TSP [16, 25]. That is, for any possible subset of released requests that might have been served by $\pi$ before $q$, we simply build two optimal paths for the parts before and after $q$ (without release times) and then we concatenate them to get a dominating permutation. We call the resulting sets *general dominating sets*. Any permutation is dominated by the one corresponding to the correct guess of requests served up to $q$. This yields a single-exponential time algorithm overall.

▶ **Theorem 4** (General Metrics). *LA-SWAG with an oracle* $\mathcal{D}$ *which uses the general dominating sets runs in single-exponential time and is* $\min\{3/2 + 5\eta, 2.75\}$*-competitive for the closed variant and* $\min\{3/2 + 5\eta, 3 - 1/6\}$*-competitive for the open variant of LA-OLTSP.*

The overarching insight behind how we reduce the runtime in specific metrics is the fact that we do not really need to try all possible subsets of requests served before $q$. For example, in trees, we first show a structural result about the optimal solutions (with release times). Specifically, we prove that the requests placed along a path from a leaf to the origin (considered the root) can be assumed to be served in a very specific order. This is the order in which the requests are encountered as one traverses the path from the leaf to the origin. This fact allows us to design a domination oracle which, roughly, provides a single permutation for any subset of *leaves* visited before $q$. Hence, we get an FPT algorithm parameterized by the number $l$ of leaves of the tree.

▶ **Theorem 5** (Trees). *There exists a Domination oracle for LA-SWAG in trees which yields a time complexity of* $O(2^l \cdot n^3)$ *for the closed variant and* $O(2^l \cdot n^4)$ *for the open variant, where* $l$ *is the number of leaves of the input tree.*

As a first step towards more general graphs, we deal with the concept of cycles by considering the ring. While we cannot retrieve the exact same structural result about online optimal solutions, we show something quite similar. Namely, we prove that the cyclic nature of the ring may be utilized only once by an optimal solution. After such a cyclic traversal, we can assume that the ring is split in half, yielding a tree which we know how to deal with.

▶ **Theorem 6** (Ring). *There exists a Domination oracle for LA-SWAG in the ring which yields a time complexity of* $O(n^3)$ *for the closed variant and* $O(n^5)$ *for the open variant.*

Finally, we combine the two previous sets of ideas to tackle flowers. Flowers are essentially comprised of a bunch of rings (petals) and a semi-line (stem), all of which are attached to the origin. It is still true that each single ring may be assumed to be traversed with a loop only once in this case. It turns out that we can consider at most 6 options for every petal.

▶ **Theorem 7** (Flowers). *There exists a Domination oracle for* `LA-SWAG` *in flowers which yields a time complexity of* $O(6^p \cdot n^3)$ *for the closed variant and* $O(6^p \cdot n^5)$ *for the open variant, where* $p$ *is the number of petals of the input flower.*

We summarize our results in Tables 1 and 2. In the closed variant, there is a lower bound of 3/2 even in the case of a line [24]. In the open one, we show a lower bound of $\approx 1.468$ even in the case of a line and there is a lower bound of 3/2 even in the case of a ring [14]. A full version of the paper including all proofs can be found in [13].

**Table 1** Consistency, smoothness, robustness and runtime guarantees of `LA-SWAG` for different metrics and variants.

| | Smoothness (Consistency for $\eta = 0$) | Robustness | | Runtime | |
|---|---|---|---|---|---|
| | Closed/Open | Closed | Open | Closed | Open |
| Tree | $3/2 + 5\eta$ | 2.5 | $3 - 1/3$ | $O(2^l \cdot n^3)$ | $O(2^l \cdot n^4)$ |
| Ring | | 2.75 | $3 - 1/6$ | $O(n^3)$ | $O(n^5)$ |
| Flower | | 2.75 | $3 - 1/6$ | $O(6^p \cdot n^3)$ | $O(6^p \cdot n^5)$ |
| Euclidean | | 2.5 | $3 - 1/6$ | $O(n^2 \cdot 2^n)$ | $O(n^2 \cdot 2^n)$ |
| General | | 2.75 | $3 - 1/6$ | $O(n^2 \cdot 2^n)$ | $O(n^2 \cdot 2^n)$ |

**Table 2** Consistency of `LA-SWAG` and other tractable algorithms. The upper bounds with * are given by an FPT algorithm and a polytime algorithm otherwise. Tight bounds are denoted in bold.

| | Consistency | | | |
|---|---|---|---|---|
| | Closed | | Open | |
| | Previous work | **This paper** | Previous work | **This paper** |
| Line | **3/2** [24] | **3/2** | 5/3 [24] | 3/2 |
| Star | 7/4+$\epsilon$ [14] | **3/2*** | 2 [14] | 3/2* |
| Tree | 2 [10] | **3/2*** | 2 [14] | 3/2* |
| Ring | 5/3 [14] | **3/2** | 2 [14] | **3/2** |
| Flower | 2 [10] | **3/2*** | 2 [14] | **3/2*** |

## 1.2 Further related works

The offline version of the problem, in which the locations and release times are known in advance, has been studied in [18, 38] for both closed and open variants. For OLTSP, a 2-competitive algorithm for the closed variant and a 2.5-competitive algorithm for the open variant have been proposed in general metric spaces by [10]. Specifically on the line, there exist lower bounds of 1.64 [10] and 2.04 [18] for the closed and open variants, respectively.

In addition to online TSP, there are several works that have explored learning augmented settings. The online caching problem with predictions was investigated by [33], and the initial results were improved by [3, 40, 42]. Adopting the LA approach, algorithms were developed for the ski-rental problem [2, 23, 39, 41] as well as for scheduling problems [4, 11, 35, 37]. There is also literature on learning augmented algorithms for classical data structures [30], bloom filters [34], routing problems [15, 22, 29], online selection and matching problems [5, 21] and a more general framework of online primal-dual algorithms [12]. There is a survey [36] and an updated list of papers [32] in this area.

## 2 Preliminaries

### Online TSP (OLTSP)

The input of OLTSP consists of a metric space $M$ with a distinguished point $\mathcal{O}$ (the origin), and a set $Q = \{q_1, ..., q_n\}$ of $n$ requests. Every request $q_i$ is a pair $(x_i, t_i)$, where $x_i$ is a point of $M$ and $t_i \geq 0$ is a real number. We use $t$ to quantify time. The number $t_i$ represents the moment after which the request $q_i$ can be served (release time). A server located at the origin at time $t = 0$, which can move with unit speed, must serve all the requests after their release times with the goal of minimizing the total completion time (makespan).

We consider a wide class of continuous metric spaces $M$ whose corresponding distance metric $d(x, y)$ is defined as the shortest path from $x \in M$ to $y \in M$ and is continuous in $M$, as in [10]. We call this class general metric spaces (or general metrics). The release times for continuous metric spaces can be any non-negative real number.

For the rest of the paper, we denote the total completion time of an online algorithm ALG by $|\,\mathrm{ALG}\,|$ and that of an optimal (offline) solution OPT by $|\,\mathrm{OPT}\,|$. We recall that an algorithm ALG is $\rho$-competitive if on all instances we have $|\,\mathrm{ALG}\,| \leq \rho \cdot |\,\mathrm{OPT}\,|$.

### Learning-augmented algorithms

In order to measure the quality of the predictions, we will define a prediction error $\eta$. LA algorithms have three main properties. We use the formal definitions in [24] here. We say that an algorithm is

- $\alpha$-*consistent*, if it is $\alpha$-*competitive* when $\eta = 0$,
- $\beta$-*robust*, if it is always $\beta$-*competitive* regardless of $\eta$, and
- $\gamma$-*smooth* for a continuous function $\gamma(\eta)$, if it is $\gamma(\eta)$-*competitive*.

In general, if $c$ is the best competitive ratio achievable without predictions, it is desirable to have $\alpha < c$, $\beta \leq k \cdot c$ for some constant $k$ and also the function $\gamma$ should increase from $\alpha$ to $\beta$ along with the error $\eta$.

### Our prediction setting

Let $Q = \{q_1, \ldots, q_n\}$ be the set of requests. As we mentioned, each request $q_i$ has a corresponding release time $t_i$ and a location $x_i$. We have a set of predictions $P = \{p_1, \ldots, p_n\}$ in which $p_i$ predicts $x_i$, the location of request $q_i$. The algorithm gets these predictions as well as the number of requests $n$ as an offline input. The actual values of $x_i$ and $t_i$ only become known at time-point $t_i$.

The predictions' quality can vary and is unknown to the algorithm. We can evaluate the quality by defining a measure $\eta$. Essentially, $\eta$ measures the sum of all the distance between the predicted location and the actual ones normalized by the length of a shortest path serving all the requests.

▶ **Definition 8** (Prediction Error). *The prediction error of an instance is defined by* $\eta = \frac{\sum_{i=1}^{n} d(x_i, p_i)}{F}$*, where* $F$ *is the length of a shortest path serving all the requests (and returning to the origin in the closed case).*

Note that the prediction error is scale invariant (i.e., it will not change if all the distances are multiplied by a constant factor), and $F$ acts as the normalization factor.

## 3 Oracle-based framework: the SWAG algorithm

In this section, we define an oracle-based algorithm, SWAG, designed for the case of perfect predictions. The oracle provides the algorithm with a set of permutations of the requests. We show that if the oracle satisfies some conditions, then SWAG has a competitive ratio of 3/2.

SWAG is actually a (slightly) modified version of the general algorithm in [14]. The principle of this latter algorithm is the following:

- First, to wait at $\mathcal{O}$ until a chosen time $T$. This time $T$ depends both on the requests' locations and on their release times.
- Then, to choose a route of serving requests that minimizes some criterion involving the length of the corresponding route and the fraction of it which is released at time $T$, and to follow this route, waiting at unreleased requests.

The calculation of the chosen time $T$ on the first step is done by computing some values on *all* the $n!$ permutations of the requests. The improvement we show here is that we can get the same competitive ratio (i.e., 3/2) while considering not the whole set of permutations, but some well chosen (and ideally small) subset that satisfies a certain property. This subset of permutations is given to the algorithm by the oracle.

Then, based on this framework, to derive an efficient 3/2-competitive algorithm, one only has to build an efficient oracle. We will show for example in Section 5.2 that for lines or rings, we can devise a polytime oracle building a polysize subset of permutations, leading to a polytime 3/2-competitive algorithm for these metrics.

More formally, we consider SWAG, which uses the following notation. For a given order $\sigma$ on the requests (where $\sigma[i]$ denotes the $i$-th request in the order), we denote:

- by $\ell_\sigma$ the length of the route associated to $\sigma$ (starting at $\mathcal{O}$), i.e., $\ell_\sigma = d(\mathcal{O}, \sigma[1]) + \sum_{j=1}^{n-1} d(\sigma[j], \sigma[j+1])$ in the open case, $\ell_\sigma = d(O, \sigma[1]) + \sum_{j=1}^{n-1} d(\sigma[j], \sigma[j+1]) + d(\sigma[n], \mathcal{O})$ in the closed case;
- by $\alpha_\sigma(t)$ the fraction of the length of the largest fully released prefix of the route associated to $\sigma$ at time $t$ over $\ell_\sigma$. More formally, if all $n$ requests are released, then $\alpha_\sigma(t) = 1$ for all $\sigma$, otherwise, if requests $\sigma[1], \ldots, \sigma[k-1]$ are released at $t$ but $\sigma[k]$ is not, then the route is fully released up to $\sigma[k]$, and

$$\alpha_\sigma(t) = \big(d(\mathcal{O}, \sigma[1]) + \sum_{j=1}^{k-1} d(\sigma[j], \sigma[j+1])\big)/\ell_\sigma \ .$$

  Note that this requires $\ell_\sigma > 0$. If $\ell_\sigma = 0$, i.e., all requests are at $\mathcal{O}$, we set $\alpha_\sigma = 1$.

The key idea behind these subsets is the following. To achieve the same competitive ratio, it is sufficient that for any possible permutation $\sigma$, the set $S$ contains a permutation $\sigma'$ that induces a tour/path that is not longer than that of $\sigma$, and its unreleased portion of the tour at time $t$ is not larger than that of $\sigma$. We will say that $\sigma'$ dominates $\sigma$ and define this notion formally below. Furthermore, since the released parts only change when a new request is released, it is sufficient to only update the subset then.

▶ **Definition 9** (Dominating Permutation). *Let $\sigma$ be a permutation of the $n$ requests and $t$ a given time. We define $Dom(\sigma, t)$ to be the set of permutations that dominate permutation $\sigma$ at time $t$. A permutation $\sigma' \in Dom(\sigma, t)$ if and only if:*

$$\ell_{\sigma'} \leq \ell_\sigma \quad \text{and} \quad (1 - \alpha_{\sigma'}(t))\ell_{\sigma'} \leq (1 - \alpha_\sigma(t))\ell_\sigma \ .$$

*We also say that $\sigma'$ is a corresponding dominating permutation of $\sigma$ (at time $t$).*

**Algorithm 1** Strategically Wait And Go (SWAG).

**Input:** Offline: request locations $x_1, \ldots, x_n$
Online: release times $t_1, \ldots, t_n$
Parameter: an oracle $\mathcal{D}$ which outputs a set of permutations on requests

Call the oracle $\mathcal{D}$ to get an initial set $S(0)$ of permutations at $t = 0$. Set $S = S(0)$.
**while** *true* **do**
  At each release time $t_i$, request a new set $S(t_i)$ of permutations and update $S = S(t_i)$. For every $\sigma \in S$, compute $\ell_\sigma$ and $\alpha_\sigma(t_i)$.
  If $\exists\, \sigma_0 \in S$ s.t. (1) $t \geq \ell_{\sigma_0}/2$ and (2) $\alpha_{\sigma_0}(t) \geq 1/2$, set $T = t$ and **break**.
**end**
At time $T$:
- Compute an order $\sigma_1$ which minimizes, over all orders $\sigma' \in S$, $(1 - \beta_{\sigma'})\ell_{\sigma'}$, where $\beta_{\sigma'} = \min\{\alpha_{\sigma'}(T), 1/2\}$.
- Follow the tour/path associated to $\sigma_1$. Serve the requests in this order, waiting at a request location if this request is not released.

We show that SWAG with an oracle $\mathcal{D}$ is 3/2-consistent if $\mathcal{D}$ is a *domination oracle*.

▶ **Definition 10** (Domination Oracle). *An oracle $\mathcal{D}$ which outputs at time $t$ a set $S(t)$ of permutations is a domination oracle if*

1. *$S(t) \subseteq S(t')$ for every $t \leq t'$, and*
2. *for all $t$ there exists a permutation $\sigma' \in S(t)$ such that $\sigma' \in Dom(\sigma_{\mathrm{OPT}}, t)$. Here, $\sigma_{\mathrm{OPT}}$ is the permutation corresponding to the serving order of requests in an optimal solution.*

▶ **Lemma 11.** *SWAG is 3/2-consistent for both closed and open variants of OLTSP with perfect predictions if it uses a domination oracle.*

Note that this matches the lower bounds in [24, 14], making the result tight in almost all cases. Regarding the computational complexity of SWAG, we have the following lemma.

▶ **Lemma 12.** *If $N$ is the maximum number of permutations which the oracle $\mathcal{D}$ outputs at each time $t$ and $T_{\mathcal{D}}$ is the total time required (by the oracle) to compute all permutations, then the running time of SWAG is $O(\max\{n^2 \cdot N, T_{\mathcal{D}}\})$.*

## 4 Performance guarantees for LA-OLTSP: the LA-SWAG algorithm

In this section, we deal with imperfect predictions; more specifically, we show how to adapt SWAG to also get smoothness and robustness upper bounds.

We note that if the predictions are perfect, then LA-SWAG is at least as good as SWAG (it works the same, the only difference being that it optimally serves the remaining unserved requests when everything is released). So in particular it is 3/2-consistent as well, provided that the oracle satisfies the conditions of Lemma 11, which is tight in almost all cases.

We also note that the algorithm could serve the requests in a more clever way: instead of going first to the predicted location of a request and then to its true location, the algorithm could go to the true location directly if it is released (or as soon as it is). However, no gain can be obtained; the tightness of analysis given in the full version of the paper would still hold.

Regarding the running time of LA-SWAG, it can be shown that if $T_{TSP}$ is the time required for the computation of an optimal path serving a subset of the requests, then from Lemma 12 we get the following corollary. Note that, for the cases we consider, we achieve suitable upper bounds on $T_{TSP}$. We describe how this is done in the full version of the paper.

**Algorithm 2** Learning-Augmented Strategically Wait And Go (LA-SWAG).

**Input:** Offline: predicted request locations $p_1, \ldots, p_n$
Online: release times $t_1, \ldots, t_n$, true request locations $x_1, \ldots, x_n$
Parameter: an oracle $\mathcal{D}$ which outputs a set of permutations on requests

(Breaking rule) At any time $t$: if all requests are released, follow an optimal path serving all unserved requests (returning to $\mathcal{O}$ if in the closed variant) and **break**.

Run SWAG until the starting time $T$ of the server, and the computation of $\sigma_1$.

At time $T$, follow the tour/path $\sigma_1$, serving the requests in the following order:
**for** $i = 1, \ldots, n$ **do**
- first go to $p_{\sigma_1[i]}$; if $q_{\sigma_1[i]}$ is not released, wait there until it is released;
- then, go to $x_{\sigma_1[i]}$ and serve the request.

**end**
In the closed version, go back to $\mathcal{O}$.

▶ **Corollary 13.** *If $N$ is the maximum number of permutations which the oracle $\mathcal{D}$ outputs at each time $t$, $T_{\mathcal{D}}$ is the total time required (by the oracle) to compute all permutations and $T_{TSP}$ is the time for the computation of an optimal path that serves a subset of the requests, then the running time of LA-SWAG is $O(\max\{n^2 \cdot N, T_{\mathcal{D}}, T_{TSP}\})$.*

## 4.1 Smoothness

In this subsection, we show that LA-SWAG is smooth with respect to the measure of error $\eta$. To show the main result, we relate the performance of LA-SWAG on a hypothetical instance using the predictions as requests. As noted above, LA-SWAG inherits its 3/2-consistency from SWAG. Leveraging triangle inequality, we further get upper and lower bounds on the performance of LA-SWAG in terms of the performance of LA-SWAG and OPT on the actual instance, respectively. This allows us to determine their maximum ratio.

▶ **Theorem 1** (Consistency and Smoothness). *LA-SWAG has a competitive ratio of at most $3/2 + 5\eta$ for both closed and open LA-OLTSP.*

Giving a simple example with 2 requests, we further show that a linear dependency on $\eta$ is necessary for the open variant for any algorithm.

▶ **Proposition 14** (Smoothness Lower Bound). *No algorithm can have a better competitive ratio than $\left(\frac{3}{2} + \frac{\eta}{2}\right)$ for the open variant on an instance with prediction error $\eta \in [0, 1/3]$.*

The example is built symmetrically, such that the adversary will be able to react to any choice made by an algorithm, thereby forcing it to take a detour. At the same time, OPT is able to quickly finish its route without having to wait or turn back.

## 4.2 Robustness

LA-SWAG can easily be shown to be 3-robust, as it can always go back to the origin when all requests are released and then follow an optimal tour. Since the distance to the origin is at most $|\,\mathrm{OPT}\,|$ when all requests are released, $3|\,\mathrm{OPT}\,|$ is an immediate upper bound. We first show improved upper bounds on the robustness of LA-SWAG, and then complement this analysis with some lower bounds.

The first step in the analysis is to assume LA-SWAG waits for some portion of $|\,\mathrm{OPT}\,|$ in the origin, say $c$. Then, a $(3-c)$ robustness follows. Otherwise, by construction of LA-SWAG, we get a suitable bound on the distances of predictions from the origin, depending on $c$.

For the following, we focus on the case where LA-SWAG leaves early. In the closed variant, we observe that the furthest away from the origin LA-SWAG can be when the last request is released is $3/4|\,\mathrm{OPT}\,|$, thereby improving the robustness bound to 2.75. This stems from the fact that the furthest prediction and the furthest request each cannot have a distance more than $1/2|\,\mathrm{OPT}\,|$ (corresponding to $c = 1/4$) from the origin as we consider the closed version, but LA-SWAG could be in between the two locations.

For trees or in a Euclidean space, the same argument can be used, but here it holds that any point in between the two locations cannot be further away from the origin than either of the two locations, which gives us overall a 2.5-robustness in these cases.

▶ **Theorem 2** (Robustness-Closed). *LA-SWAG is 2.75-robust for closed LA-OLTSP in general metric spaces, and 2.5-robust in Euclidean spaces and in trees.*

In the open variant, we can use a similar approach. While going back to the origin and following OPT is always a viable option when everything is released, we can also go the the final request OPT visited and then take OPT backwards to serve all requests. It turns out that these two choices are sufficient to improve the robustness below 3. We again consider general metrics first and show an improvement to $3 - 1/6$, before we again study the case of trees to show that a bigger improvement to $3 - 1/3$ is possible in this scenario.

▶ **Theorem 3** (Robustness-Open). *LA-SWAG is $(3 - 1/6)$-robust for open LA-OLTSP in general metric spaces, and $(3 - 1/3)$-robust in trees.*

We show in the full version that the analyses for 2.5-robustness in the closed variant and $(3 - 1/3)$-robustness in the open variant are tight. Moreover, we show the following consistency-robustness tradeoff for any algorithm in the open variant. For a good consistency result, the algorithm has to trust the predictions more, which will lead to a bad robustness.

▶ **Proposition 15.** *Let* ALG *be an algorithm for the open variant with consistency guarantee $2 - \lambda$, where $\lambda \in [0, 1]$. Then,* ALG *cannot be $(2 + \lambda - \epsilon)$-robust, for any $\epsilon > 0$.*

## 5 Domination oracles

In this section, we implement a domination oracle for general metrics using a Dynamic Programming inspired technique and then we exploit structural insights about specific metrics to obtain more efficient oracles. We focus on the closed variant; we explain in the full version how to deal with the open variant. In the following, $R(t)$ and $U(t)$ will refer to the released and unreleased requests at time $t$. We may drop the dependency on $t$.

### 5.1 General metrics

In the case of perfect predictions, using SWAG and Lemma 11 we can reduce the factorial running time of the algorithm in [14] for general metrics. We will show that an exponential number of permutations suffices to have a domination oracle. After that, using the scheme of LA-SWAG and the theorems proved in the previous section we will get an exponential-time algorithm that is 3/2-consistent, smooth and robust.

Consider an arbitrary permutation $\sigma$ of the $n$ requests at time $t$. A corresponding dominating tour for this permutation can be constructed in the following way; find the first unreleased request $u \in U(t)$ in the permutation $\sigma$. The released requests before $u$ in

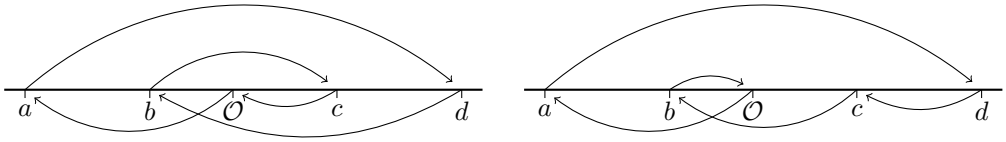



**Figure 1** Two permutations illustrating the notion of domination as in Example 16. Left side: OPT, right side: $\sigma'$.

the permutation form a subset $R'(t) \subseteq R(t)$ of released requests. Then, a corresponding general dominating tour would be the optimal TSP path that starts from the origin, serves all requests in $R'(t)$ and ends at request $u$ plus the optimal TSP path that starts from $u$, serves all requests in $\big(R(t) \setminus R'(t)\big) \cup \big(U(t) \setminus \{u\}\big)$ and returns to the origin. We will call these tours (corresponding) *general dominating tours*. It can be shown that at time $t$ a corresponding general dominating tour $\sigma'$ of a permutation $\sigma$ dominates $\sigma$. In Example 16, we illustrate the notion of a dominating tour.

▶ **Example 16.** Consider the following example on the line in the closed variant. There are 4 requests $a, b, c, d$, located at $-10, -4, 5, 11$, respectively. Assume the release times to be $t_a = 10, t_b = 46, t_c = 55, t_d = 31$. The optimal tour OPT is $\mathcal{O} \to a \to d \to b \to c \to \mathcal{O}$ with $|\operatorname{OPT}| = 60$, never having to wait and reaching requests just as they are released. Thus, we also have $\ell_{\operatorname{OPT}} = 60$.

Let us focus on the setting in which only the first request has been released, i.e., for $t \in [10, 31)$. At this time, we have $R(t) = \{a\}$ and $U(t) = \{b, c, d\}$. We can see that $d$ is the first unreleased request in permutation OPT at this time, so $u = d$, and $R' = R$. The first part of the general dominating permutation $\sigma'$ of OPT is thus $\mathcal{O} \to a \to d$, and the second part $d \to c \to b \to \mathcal{O}$. This means that $\ell_{\sigma'} = 50 < 60 = \ell_\sigma$. An illustration of OPT and $\sigma'$ is given in Figure 1.

When we run SWAG, we can see why this notion of domination makes sense: Clearly, the released lengths of OPT and $\sigma'$ are 31 for $t \in [10, 31)$ (for both permutations, the server can already serve $a$ and continue to $d$ without having to wait anywhere – but $d$ itself cannot yet be served). This also means that $\alpha_{\operatorname{OPT}} \geq 1/2$ and $\alpha_{\sigma'} \geq 1/2$. Thus, both of the permutations would start the second part of the algorithm when $t$ would be large enough. Since $\ell_{\sigma'} \leq \ell_\sigma$, it suffices to consider the permutation $\sigma'$ instead of $\sigma$, as regardless of the release time of $a$, no earlier starting time could be computed when using OPT than when using $\sigma'$. Additionally, $\beta_{\sigma'}(t) = \beta_{\operatorname{OPT}}(t) = 1/2$ and thus OPT would not be chosen as permutation to follow in the second part of the algorithm.

Note that, if $a, b$, and $d$ are released but $c$ is not at some time $t'$, then OPT would not be dominated at $t'$ by $\sigma'$. This is because the unreleased part of $\sigma'$ would be greater than that of OPT. Clearly, OPT would be dominated at time $t'$ by another permutation, e.g., $\mathcal{O} \to a \to b \to d \to c \to \mathcal{O}$. ◀

If we assume that $t_1 \leq t_2 \leq \cdots \leq t_n$ are the release times of the requests, $t_0 = 0$ and $t_{n+1} = \infty$, then at time $t_i$, with $i \leq n$, we compute the set of general dominating permutations $D_i$ given the sets $R(t_i)$ and $U(t_i)$. Together with the previously computed sets, it is then provided to SWAG as set $S(t_i) = \bigcup_{j=0}^{i} D_j$. It is easy to show that an oracle $\mathcal{D}$ which outputs at any time $t \in [t_i, t_{i+1})$ the general dominating sets $S(t) = \bigcup_{j=0}^{i} D_j$ for each $i \in \{0, 1, 2, \ldots, n\}$, is a domination oracle and SWAG is 3/2-consistent (Lemma 11). Condition 1 of the domination oracle is satisfied as $S(t) \subseteq S(t')$ for every $t \leq t'$. Condition 2 is also satisfied since for each time $t$ we have a corresponding dominating permutation for every possible subset of $R(t)$ and choice of $u$, and thus for every permutation $\sigma$.

Concerning the running time of the algorithm, we begin by bounding the number of general dominating permutations which are contained in the set $\bigcup_{j=0}^{i} D_j$ for all $i \leq n$. Formally, we prove the following lemma.

▶ **Lemma 17.** *There always exists a set $\bigcup_{j=0}^{i} D_j$ which consists of at most $2^n$ distinct general dominating permutations, for all $i \in \{0, 1, \ldots, n\}$.*

In order to get an LA algorithm, we run `LA-SWAG` with the oracle of general dominating permutations. Since all possible general dominating permutations can be precomputed at $t = 0$ using the dynamic programming algorithm for classical TSP in time $O(n^2 \cdot 2^n)$, we use Corollary 13 and Lemma 17 to show the following result for the running time of `LA-SWAG`.

▶ **Lemma 18.** *`LA-SWAG` that uses the general dominating sets runs in $O(n^2 \cdot 2^n)$ time complexity for both closed and open variants of OLTSP.*

Using theorems 1, 2, and 3 we show that `LA-SWAG` with the general dominating permutations is a single-exponential time algorithm with the following performance guarantees.

▶ **Theorem 4** (General Metrics). *`LA-SWAG` with an oracle $\mathcal{D}$ which uses the general dominating sets runs in single-exponential time and is $\min\{3/2 + 5\eta, 2.75\}$-competitive for the closed variant and $\min\{3/2 + 5\eta, 3 - 1/6\}$-competitive for the open variant of LA-OLTSP.*

## 5.2 Specific metrics

In the following we explain how to implement domination oracles for trees, rings, and flowers. These will dominate a so-called *sensible* set of permutations, which we define later.

We also need to implement the "cleanup" step of `LA-SWAG`, where a computation of an optimal classical (offline, without release times) TSP path/tour is required. In fact, these subroutines are also employed within the domination oracles themselves. We describe how this is done efficiently (in $O(n)$ time) in the full version of the paper.

▶ **Definition 19** (Safe Set). *Let $X$ be a set of request locations. A set of permutations $\Pi$ is safe for $X$ if for any assignment $f$ of release times to each location in $X$, there exists a permutation $\pi \in \Pi$ which is optimal for the resulting input $Q = \{(x, f(x)) \mid x \in X\}$.*

Note that the set of all permutations is safe. However, we can define *sensible* sets of permutations that follow some desirable structure but are still safe. The guarantees of `LA-SWAG` follow even when dominating a sensible set.

In general, the idea is to ensure domination as follows. Let $Q_1$ be the requests served by a permutation $\pi$ before its current request $q$. We construct a permutation $\pi_{dom}$, which serves a superset of $Q_1$ before $q$ and is not longer than $\pi$ on both parts, before and after $q$.

### Trees

We consider metric spaces that are shaped like a tree. By that we mean continuous spaces that can be embedded into trees with edges weighted by their lengths. The requests may appear on nodes of the tree or at some point along the edges.

Suppose that an algorithm serves a request $q$ at time $t$. There is only one path $\mathcal{P}_q$ from $q$ to $\mathcal{O}$, and it must be traversed after $t$. Hence, any request $q'$ on $\mathcal{P}_q$ will be visited at some point after $t$ anyway, rendering the earlier serving of such requests useless. From this, it follows that any sequence of requests $S_q$ encountered along a path $\mathcal{P}_q$ towards the origin may be safely assumed to be a subsequence of the optimal permutation.

▶ **Definition 20** (Sensible Set for Trees). *Let $\mathcal{T}$ be a tree rooted at the origin $\mathcal{O}$ and $X$ a set of request locations on $\mathcal{T}$. The set $\Pi_s(\mathcal{T}, X)$ of sensible permutations consists of all permutations $\pi$ where the following holds. Let $\pi = x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$. For any $x_{\pi(i)}$ and any $x_{\pi(j)}$ on the path $\mathcal{P}_{x_{\pi(i)}}$ from $x_{\pi(i)}$ to the origin, we have $j > i$.*

▷ Claim 21. For any tree $\mathcal{T}$ and any set of request locations $X$, the set $\Pi_s(\mathcal{T}, X)$ is safe.

### Dominating set

For the above sensible set, we will define a dominating set whose cardinality is of order $O(2^l \cdot n)$, where $l$ is the number of leaves of the underlying tree. The idea is to look at subsets of paths of leaves to the origin. That is, for each such subset and each selection of intermediate request $q$, we consider the permutation which first optimally visits the selected leaves finishing at $q$ and then "cleans up" the rest of the requests starting from $q$ and ending at the origin. The set of these permutations for all different choices of leaf subsets and current request is called $Dom^{\curlyvee}(R, U, \mathcal{T}, L)$, where $\mathcal{T}$ is the input tree and $L$ is its set of leaves.

▶ **Lemma 22** (Tree Domination). *Let $\pi$ be a sensible permutation for a tree $\mathcal{T}$ and $L$ be its set of leaves. Then, $\exists\, \pi_{dom} \in Dom^{\curlyvee}(R, U, \mathcal{T}, L)$ dominating $\pi$.*

Using the above lemma, we obtain our main theorem for trees.

▶ **Theorem 5** (Trees). *There exists a Domination oracle for* `LA-SWAG` *in trees which yields a time complexity of $O(2^l \cdot n^3)$ for the closed variant and $O(2^l \cdot n^4)$ for the open variant, where $l$ is the number of leaves of the input tree.*

## Ring

If a permutation makes more than one "full" loops around the ring, it may as well keep only the last. Moreover, no serving needs to be done before the loop. After such a loop, we can assume that the permutation moves along the ring as if it were a line, i.e., it never crosses the antipodal point of the origin. Hence, we can define the sensible permutations by branching on whether a loop is performed and then reduce to the tree case.

▶ **Definition 23** (Sensible Set for the Ring). *Let $X$ be a set of request locations on the ring. The sensible set of permutations $\Pi_s^{\circ}(X)$ consists of all permutations $\pi$ resulting from the concatenation of $\pi_{loop}$ and $\pi_{line}$ where $\pi_{loop}$ is a subpermutation covering $X' \subseteq X$ in a cyclic fashion and $\pi_{line} \in \Pi_s(T, X \setminus X')$, where $T$ is the tree resulting from splitting the ring at the midpoint across from the origin.*

▷ Claim 24. For any set of request locations $X$, the set $\Pi_s^{\circ}(X)$ is safe.

### Dominating set

It suffices to define the dominators under the assumption that $\pi$ does indeed have a loop in the beginning, since we can take the union with the tree dominators.

A crucial distinction is whether the current request $q$ of $\pi$ is part of the loop or not. In the first case, $\pi$ has only traveled a portion of the loop so far and in the second case it has traveled a full loop and serves as if on a tree afterwards. For every choice of $q$, we define two permutations, one pertaining to the first case and another to the second one.

The first permutation serves all released requests within $\pi$'s traveled portion of the loop and then moves to $q$. The second permutation serves all released requests with a full loop and then moves to $q$. Both finish by cleaning up the rest of the requests. We call the resulting set of permutations $Dom^{\circ}(R, U)$.

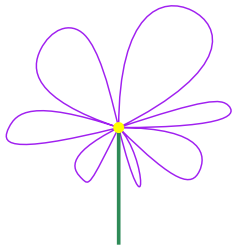


**Figure 2** A flower with 7 petals (purple), connected with the stem (green) at the origin (yellow).

▶ **Lemma 25** (Ring Domination). *Let $\pi$ be a sensible permutation whose tour contains a full loop in the beginning. $Dom^{\circ}(R, U)$ contains a permutation $\pi_{dom}$ which dominates $\pi$.*

Thus, our main theorem for the ring follows.

▶ **Theorem 6** (Ring). *There exists a Domination oracle for* `LA-SWAG` *in the ring which yields a time complexity of $O(n^3)$ for the closed variant and $O(n^5)$ for the open variant.*

## Flowers

We extend the reasoning used for the ring and trees to the so-called flowers. A flower consists of a number of rings (petals), all of which are attached to the origin point (receptacle). For reasons of artistic completion, a semi-line (stem) disjoint from the petals is also attached to the origin point. An illustration is given in Figure 2.

### Sensible set

We can assume that any sensible permutation traverses a petal of the flower in a cyclic fashion at most once, for the same reason that this is true for the ring. Thus, the sensible set here is defined to include the permutations which have at most one loop for each petal, such loops are the first visit on the corresponding petal and the restrictions of the sensible set on trees also apply to every petal after such a loop is carried out.

### Dominating set

First, choose the current request $q$ of the permutation $\pi$ to be dominated. Then, we guess the subset $\mathcal{P}$ of petals "looped" by $\pi$. We also guess the petals $\mathcal{P}_{done} \subseteq \mathcal{P}$ already looped before $q$. For a set of such choices, we traverse all petals in $\mathcal{P}_{done}$ in an arbitrary order. Next, we "snip" every petal except the ones in $\mathcal{P}$, yielding a tree $\mathcal{T}$ (actually, a star) and the petals in $\mathcal{P}$.

Now, we also guess the subset $L'$ of the leaves $L$ of $\mathcal{T}$ to be visited before $q$. Having fixed that, we append to our permutation the optimal path from $\mathcal{O}$ to $q$ that visits every leaf in $L'$ and the possible arc of $\pi$ to $q$. Finally, we append the optimal path from $q$ that "cleans up" the remaining unserved requests. The resulting set is called $Dom^{✿}(R, U)$.

▶ **Lemma 26** (Flower Domination). *Let $\pi$ be a sensible permutation for the flower. The set $Dom^{✿}(R, U)$ contains a permutation $\pi_{dom}$ which dominates $\pi$.*

Note that to make all these different guesses, one has at most 6 choices for every petal. So, if the graph has $p$ petals, the cardinality of the dominators is $O(6^p \cdot n) = O(2^{2.59p} \cdot n)$. The below complexity comes from the term related to the cardinality of the dominating set.

▶ **Theorem 7** (Flowers). *There exists a Domination oracle for* `LA-SWAG` *in flowers which yields a time complexity of* $O(6^p \cdot n^3)$ *for the closed variant and* $O(6^p \cdot n^5)$ *for the open variant, where* $p$ *is the number of petals of the input flower.*

## 6 Conclusion

We studied Online TSP augmented with predictions regarding the locations of the requests. Our algorithm, `LA-SWAG`, achieves a competitive ratio of 3/2 under the assumption of perfect predictions, which is tight in most cases we considered. Additionally, it is smooth and provides robustness guarantees below 3, improving over previous work. The runtime of `LA-SWAG` is single-exponential; however, we show how to remove the exponential dependency on the number of requests for specific metric spaces.

We believe that our techniques can be generalized to obtain FPT algorithms for other classes of graphs also; cactus graphs, graphs of bounded treewidth in general, as well as planar graphs are interesting options. Additionally, extending the algorithm to the Dial-A-Ride Problem seems like a reasonable direction to follow.

Another interesting direction is to leverage the ideas behind the PTAS for classical TSP on the Euclidean plane [6, 7] (or any other space which admits an approximation scheme for that matter) to obtain consistency guarantees which approach 3/2 arbitrarily close as the computational time is allowed to increase. Note that, since our smoothness proof follows from consistency itself and robustness from solving a classical TSP instance, these results would automatically extend, modulo some worsening of the bounds as a function of the approximation quality. For this, one possible approach is to extend the notion of a sensible set to that of $\epsilon$-sensible, meaning that a $(1+\epsilon)$-approximation of the optimal Online TSP solution is guaranteed to exist within the set. A similar relaxation would also make sense for the idea of dominating sets.

## References


1. Luca Allulli, Giorgio Ausiello, and Luigi Laura. On the power of lookahead in on-line vehicle routing problems. In *COCOON*, 2005.
2. Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *ITCS*, 2020.
3. Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *ICML*, 2020.
4. Antonios Antoniadis, Peyman Jabbarzade Ganje, and Golnoosh Shahkarami. A novel prediction setup for online speed-scaling. *CoRR abs/2112.03082*, 2021. `arXiv:2112.03082`.
5. Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In *NeurIPS*, 2020.
6. Sanjeev Arora. Polynomial time approximation schemes for euclidean TSP and other geometric problems. In *FOCS*, 1996.
7. Sanjeev Arora. Nearly linear time approximation schemes for euclidean TSP and other geometric problems. In *FOCS*, 1997.
8. Norbert Ascheuer, Martin Grötschel, Sven O. Krumke, and Jörg Rambau. Combinatorial online optimization. In Peter Kall and Hans-Jakob Lüthi, editors, *Operations Research Proceedings 1998*, pages 21–37, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
9. Giorgio Ausiello, Marc Demange, Luigi Laura, and Vangelis Paschos. Algorithms for the on-line quota traveling salesman problem. *Inf. Process. Lett.*, 92(2):89–94, 2004.
10. Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29:560–581, 2001.

11 Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *NeurIPS*, 2020.

12 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *NeurIPS*, 2020.

13 Evripidis Bampis, Bruno Escoffier, Themis Gouleakis, Niklas Hahn, Kostas Lakis, Golnoosh Shahkarami, and Michalis Xefteris. Learning-augmented online TSP on rings, trees, flowers and (almost) everywhere else. *CoRR*, abs/2305.02169, 2023. `arXiv:2305.02169`.

14 Evripidis Bampis, Bruno Escoffier, Niklas Hahn, and Michalis Xefteris. Online TSP with Known Locations. *To appear in WADS*, 2023. CoRR abs/2210.14722.

15 Evripidis Bampis, Bruno Escoffier, and Michalis Xefteris. Canadian traveller problem with predictions. In Parinya Chalermsook and Bundit Laekhanukit, editors, *Approximation and Online Algorithms*, pages 116–133, Cham, 2022. Springer International Publishing.

16 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, January 1962.

17 Giulia Bernardini, Alexander Lindermayr, Alberto Marchetti-Spaccamela, Nicole Megow, Leen Stougie, and Michelle Sweering. A universal error measure for input predictions applied to online graph problems. In *NeurIPS*, 2022.

18 Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online TSP on the line. *ACM Trans. Algorithms*, 17(1):3:1–3:58, 2021.

19 Michiel Blom, Sven O. Krumke, Willem E. De Paepe, and Leen Stougie. The online TSP against fair adversaries. *INFORMS J. Comput.*, 13(2):138–148, 2001.

20 Shuchi Chawla and Dimitris Christou. Online time-windows TSP with predictions. *CoRR*, abs/2304.01958, 2023. `arXiv:2304.01958`.

21 Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. In *EC*, 2021.

22 Franziska Eberle, Alexander Lindermayr, Nicole Megow, Lukas Nölke, and Jens Schlöter. Robustification of online graph exploration methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9732–9740, June 2022.

23 Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *ICML*, 2019.

24 Themistoklis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. Learning-Augmented Algorithms for Online TSP on the Line. In *AAAI*, 2023.

25 Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10(1):196–210, 1962.

26 Hsiao-Yu Hu, Hao-Ting Wei, Meng-Hsi Li, Kai-Min Chung, and Chung-Shou Liao. Online TSP with predictions. *CoRR*, abs/2206.15364, 2022. `arXiv:2206.15364`.

27 Patrick Jaillet and Michael R. Wagner. Online routing problems: Value of advanced information as improved competitive ratios. *Transportation Science*, 40:200–210, May 2006.

28 Vinay A. Jawgal, V. N. Muralidhara, and P. S. Srinivasan. Online travelling salesman problem on a circle. In T.V. Gopal and Junzo Watada, editors, *Theory and Applications of Models of Computation*, pages 325–336, Cham, 2019. Springer International Publishing.

29 Murali Kodialam and T. V. Lakshman. Prediction augmented segment routing. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6, 2021.

30 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *SIGMOD*, 2018.

31 Eugene L. Lawler, Jan K. Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley Series in Discrete Mathematics & Optimization, 1991.

32 Alexander Lindermayr and Nicole Megow. Alps. `https://algorithms-with-predictions.github.io/`.

**33** Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021. doi:10.1145/3447579.

**34** Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *NeurIPS*, 2018.

**35** Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *ITCS*, 2020.

**36** Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020.

**37** Benjamin Moseley, Sergei Vassilvitskii, Silvio Lattanzi, and Thomas Lavastida. Online scheduling via learned weights. In *SODA*, 2020.

**38** Harilaos N. Psaraftis, Marius M. Solomon, Thomas L. Magnanti, and Tai-Up Kim. Routing and scheduling on a shoreline with release times. *Manag. Sci.*, 36(2):212–223, 1990.

**39** Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *NeurIPS*, 2018.

**40** Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *SODA*, 2020.

**41** Shufan Wang and Jian Li. Online algorithms for multi-shop ski rental with machine learned predictions. In *AAMAS*, 2020.

**42** Alexander Wei. Better and simpler learning-augmented online caching. In *APPROX/RANDOM*, 2020.

# A Parameterized Algorithm for Vertex Connectivity Survivable Network Design Problem with Uniform Demands

**Jørgen Bang-Jensen**
University of Southern Denmark, Odense, Denmark

**Kristine Vitting Klinkby**
University of Southern Denmark, Odense, Denmark

**Pranabendu Misra**
Chennai Mathematical Institute, India

**Saket Saurabh**
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

## Abstract

In the Vertex Connectivity Survivable Network Design (VC-SNDP) problem, the input is a graph $G$ and a function $d : V(G) \times V(G) \to \mathbb{N}$ that encodes the vertex-connectivity demands between pairs of vertices. The objective is to find the smallest subgraph $H$ of $G$ that satisfies all these demands. It is a well-studied NP-complete problem that generalizes several network design problems. We consider the case of *uniform demands*, where for every vertex pair $(u, v)$ the connectivity demand $d(u, v)$ is a fixed integer $\kappa$. It is an important problem with wide applications.

We study this problem in the realm of Parameterized Complexity. In this setting, in addition to $G$ and $d$ we are given an integer $\ell$ as the *parameter* and the objective is to determine if we can remove at least $\ell$ edges from $G$ without violating any connectivity constraints. This was posed as an open problem by Bang-Jansen et.al. [SODA 2018], who studied the edge-connectivity variant of the problem under the same settings. Using a powerful classification result of Lokshtanov et al. [ICALP 2018], Gutin et al. [JCSS 2019] recently showed that this problem admits a (non-uniform) FPT algorithm where the running time was unspecified. Further they also gave an (uniform) FPT algorithm for the case of $\kappa = 2$. In this paper we present a (uniform) FPT algorithm any $\kappa$ that runs in time $2^{O(\kappa^2 \ell^4 \log \ell)} \cdot |V(G)|^{O(1)}$.

Our algorithm is built upon new insights on vertex connectivity in graphs. Our main conceptual contribution is a novel graph decomposition called the *Wheel decomposition*. Informally, it is a partition of the edge set of a graph $G$, $E(G) = X_1 \cup X_2 \ldots \cup X_r$, with the parts arranged in a cyclic order, such that each vertex $v \in V(G)$ either has edges in at most two consecutive parts, or has edges in every part of this partition. The first kind of vertices can be thought of as the rim of the wheel, while the second kind form the hub. Additionally, the vertex cuts induced by these edge-sets in $G$ have highly symmetric properties. Our main technical result, informally speaking, establishes that "nearly edge-minimal" $\kappa$-vertex connected graphs admit a wheel decomposition – a fact that can be exploited for designing algorithms. We believe that this decomposition is of independent interest and it could be a useful tool in resolving other open problems.

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

**Keywords and phrases** Parameterized Complexity, Vertex Connectivity, Network Design

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.13

**Funding** *Pranabendu Misra*: Supported by Google India Research Award 2022, and Start-Up Grant 2022 (SRG/2022/001927) of Science and Engineering Research Board (SERB), India.
*Saket Saurabh*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416), and Swarnajayanti Fellowship (No. DST/SJF/MSA01/2017-18).

## 1 Introduction

One of the most important challenges in designing real world networks is to ensure their reliability in the face of damages and equipment failures. A natural solution is to build additional redundancy in the network, at lowest possible costs, to guarantee connectivity up to a certain number of failures. This motivates the study of *network design problems*, and they are well studied in combinatorial optimization and algorithm design. The problem is abstractly described using graphs, where vertices naturally represent the nodes of the network, the edges represent the connections between the nodes and each edge has an associated cost. In many applications, the objective is to find a minimum cost subgraph that connects all the nodes and is also able to withstand a certain number of failures of vertices or edges. We refer to the surveys [14, 19, 22] for details.

Formally, in the case of vertex failures this problem is called Minimum $\kappa$-Vertex Connected Spanning Subgraph ($\kappa$-VCSS); the input is a graph $G$ on $n$ vertices with edge costs $w : E(G) \to \mathbb{R}^+$, and the objective is to find a spanning subgraph $H$ of minimum total cost such that it remains connected even when $\kappa - 1$ vertices are deleted, for some fixed integer $\kappa$. And in the case of edge-failures, we get Minimum $\lambda$-Edge Connected Spanning Subgraph ($\lambda$-ECSS) where we must ensure network connectivity even after $\lambda - 1$ edges fail, for some fixed integer $\lambda$. Both these problems are very well studied, and have wide applications. Unfortunately they are NP-hard, even in the unweighted setting. Therefore they have been extensively studied in Approximation algorithms [14, 19, 22], and more recently in Parameterized algorithms.

If a pair of vertices $u, v \in V(G)$ remain connected even after $\kappa - 1$ vertex failures, then by the Menger's Theorem [3, Theorem 7.3.1], there must be $\kappa$ internally disjoint paths from $u$ to $v$ in $G$. In other words, the *vertex-connectivity between $u$ and $v$* is at least $\kappa$ in $G$. Since every pair of vertices in $G$ must have this property, the graph $G$ must be $\kappa$-vertex connected. Similarly, in the case of edge-connectivity, the graph $G$ must be $\lambda$-edge connected. Interpreting $\kappa$-VCSS / $\lambda$-ECSS in terms of connectivity leads to the Survivable Network Design Problem (SNDP), where we may have different connectivity demands between different vertex pairs. SNDP captures a number of network design problems such as Steiner Tree, Minimum Equivalent Digraph, Hamiltonian Cycle etc. We can classify many network design problems by the nature of their connectivity constraints into variants of Edge Connectivity SNDP (EC-SNDP) and Vertex Connectivity SNDP (VC-SNDP). Observe that $\kappa$-VCSS and $\lambda$-ECSS correspond to the uniform connectivity demands case of these problems, respectively. Hence they are also called VC-SNDP / EC-SNDP with Uniform Demands. There has been a vast amount on research network design problems, especially in Approximation algorithms [14, 19, 22], which has led to the development of a number of new algorithmic techniques; e.g. the 2-approximation algorithm of Jain for EC-SNDP [13] which introduced iterative LP-rounding.

Typically, vertex connectivity problems are often significantly more difficult than the corresponding edge-connectivity problems. In contrast to the 2-approximation for EC-SNDP, VC-SNDP has an approximation lower bound of $2^{\log^{1-\epsilon} n}$ [18], and $\kappa^\epsilon$ for every $\kappa > \kappa_0$ where $\epsilon > 0$ and $\kappa_0 > 1$ are constants [6]. Even in the case of uniform requirements, i.e. $\kappa$-VCSS, the best known approximation algorithm for $\kappa$-VCSS has an approximation factor $\mathcal{O}(\log \kappa \cdot \log \frac{n}{n-\kappa})$[23], although it is improved to 6 assuming that $n \geq \kappa^3(\kappa - 1) + \kappa$ [7]. Another example is the Unrestricted Vertex Connectivity Augmentation (UVCA), where the objective is to augment a $\kappa$-vertex connected graph to a $\kappa + t$-vertex connected

graph by adding new edges.[1] While Unrestricted Edge Connectivity Augmentation (UECA), that is similarly defined, is known to be in polynomial time from many decades ago[11], the complexity of UVCA still remains open. Currently, a polynomial time algorithm is only known for $t = 1$ [26].

Recently, there has been a lot of interest in the study of the Parameterized algorithms [8][2] for network design problems [2, 21, 4, 1, 12, 15, 16, 17, 10, 9]. The separation between vertex-connectivity problems and edge-connectivity problems also persists in this setting; nearly all of the currently known results are for edge-connectivity problems. Bang-Jensen et al. [1] gave an FPT algorithm for the parameterized version of $\lambda$-ECSS[3] in this setting. This naturally raises the question for $\kappa$-VCSS, denoted by p-$\kappa$-VCSS[4], which they pose as an open problem. This is the problem we study here. Gutin et.al. [12] gave a *non-uniform FPT algorithm*[5] for p-$\kappa$-VCSS using a powerful classification result of Lokshtanov et al. [20]. The running time of this algorithm is unspecified. They also gave an (uniform) FPT algorithm for p-$\kappa$-VCSS when $\kappa = 2$ (Biconnectivity Deletion) [12].

| p-$\kappa$-VCSS | **Parameter:** $\ell$ |
|---|---|
| **Input:** A $\kappa$-connected graph $G = (V, E)$ and an integer $\ell$. | |
| **Question:** Does there exist a set of edges $F \subseteq E$ such that $\lvert F\rvert \geq \ell$ and $G' = (V, E \setminus F)$ is $\kappa$-connected? | |

**Our contribution.** In this work we give an (uniform) FPT algorithm for p-$\kappa$-VCSS for any constant $\kappa$; indeed our result is stronger and the algorithm is FPT in both $\ell$ and $\kappa$. We build upon the broad approach of Bang-Jensen et al. [1], but develop new insights into vertex connectivity that are of independent interest.

▶ **Theorem 1.** p-$\kappa$-*VCSS admits an FPT algorithm running in time* $2^{\mathcal{O}(\kappa^2\ell^4 \log \ell)} \cdot |G|^{\mathcal{O}(1)}$.

Our algorithm follows the general scheme laid down by Bang-Jensen et al. [1] for p-$\lambda$-ECSS. We consider the set of all *deletable edges*, i.e. all those edges $e \in G(G)$ such that $G - e$ is $\kappa$-vertex connected. Our goal is to identify an *irrelevant edge*[6] among these edges and shrink the pool of *relevant* deletable edges until the instance can be more easily solved. Bang-Jensen et al. [1] show that, for p-$\lambda$-ECSS, if the graph contains a large number of deletable edges, then there exists a cyclic decomposition of the graph that can be used to identify an irrelevant edge.[7] We prove a similar kind of result for p-$\kappa$-VCSS.

We develop a new tool called the *Wheel decomposition* for this purpose that helps us understand the structure of "nearly edge minimal" $\kappa$-vertex connected graphs. This is our main conceptual contribution. An intuitive description is as follows: It is a partition of the

[1] Formally, the input is a $\kappa$-vertex connected graph $G$ and an integer $t \geq 1$. The objective is to compute a minimum cost subset $F \subseteq V \times V$ such that $G + F$ is $(\kappa + t)$-vertex connected.

[2] Let us recall that in Parameterized Complexity, the input is a pair $(X, \ell)$ where $X$ is an instance of the problem and $\ell$ is an integer, called the *parameter*. The objective is to design a *Fixed Parameter Tractable (FPT) algorithm*, i.e. an algorithm that solves the problem in time $f(\ell) \cdot |X|^{\mathcal{O}(1)}$ where $f$ is a function of $\ell$ alone.

[3] They give FPT algorithms for $\lambda$-ECSS in both graphs and digraphs. Further, they extended their results to $\kappa$-VCSS in digraphs by reducing it to special instances of $\lambda$-ECSS.

[4] Here the $p$ in p-$\kappa$-VCSS denotes *parameterized*

[5] Here *non-uniform algorithm* refers to the complexity theory term, which is different from demands being uniform or non-uniform.

[6] i.e. one that is not present in some solution to the input instance and therefore remains in the graph when the edges of that solution are deleted.

[7] This is specifically required for odd values of $\lambda$ in undirected graphs. The other cases are much simpler.

edge set $E(G)$ of a graph $G$, with the parts arranged in a cyclic order. Any vertex either has edges in at most two consecutive parts, or else it has edges in every part of the partition.[8] Each of these parts, which are edge-subsets, describes a vertex cut in a natural way: the set of all those vertices that have an edge in this part and another edge somewhere outside it. In a Wheel decomposition, these vertex cuts are highly symmetric. For our results, we develop algorithmic tools to construct Wheel decompositions in instances of p-$\kappa$-VCSS and then identify irrelevant edges using them.

We believe that the Wheel decomposition is of independent interest. As we have observed earlier, vertex connectivity problems are often substantially more difficult than the corresponding edge-connectivity problems. One reason for this is the lack of structural results and tools for vertex connectivity as compared to edge connectivity. We believe that Wheel decompositions will be a useful tool in understanding vertex connectivity and resolving other open questions in the future.

**Related works.** As we mention above, our work has been most influenced by the work of Bang-Jensen et al. [1]; we however require several new ideas and methods to deal with vertex connectivity. Other related works gave FPT algorithms for Directed and Undirected Spanners [16, 17, 10], and 2-VCSS [12] in a similar setting. A recent work considered these problems parameterized by the solution size [9]. Additional results are known about connectivity augmentation problems, e.g. Strong Connectivity Augmentation [15], Edge connectivity Augmentation by One [21, 4] and Minimum Strong Spanning Subgraph [2].There is a vast amount of research on network design problems in approximation algorithms, too many to list comprehensively. We refer to the following surveys for an overview [14, 19, 22]. There has been recent interest in improving the approximation-factor for some specific simple cases of EC-SNDP, such as Weighted Tree Augmentation and Weighted Connectivity Augmentation by One). A sequence of works have finally led to breaching the approximation factor barrier 2 for both these problems [5, 24, 25]. Ideas and methods from the Parameterized algorithms for these problems also played a role in some of these results [5, 4].

## 2 Preliminaries and Notation

For a universe $W$ and a subset of elements $U \subseteq W$ we will denote the set $W \setminus U$ by $\overline{U}$. When doing set-operations where one of the sets is a singleton, e.g. $A \cup \{x\}$, we omit the curly-braces and just write $A \cup x$. For a graph $G$ and a subset of vertices $A \subseteq V(G)$, $G - A$ denotes the induced subgraph $G[V(G) \setminus A]$. Similarly, for a set of edges $B \subseteq E(G)$, $G - B$ denotes the subgraph with vertex set $V(G)$ and edge-set $E(G) \setminus B$. For a graph $G$ and a subset of vertex-pairs $B \subseteq V \times V$, $G + B$ denotes the graph with vertex set $V(G)$ and edge-set $E(G) \cup B$. Let $G = (V, E)$ be a graph and for an edge $e = uv \in E$ the **endpoints of e** are the vertices $u$ and $v$, and we have $V(e) = \{u, v\}$. For a vertex $u \in V$ let $\delta(u) \in E$ define the set of edges which have $u$ as an endpoint. We extend this notion to subsets of vertices, i.e. $\delta(X)$ denotes the set of edges with an endpoint in $X$. Further, for an edge $e = (u, v)$, $\delta(e)$ denotes the set $\delta(V(e))$. For a set of edges $E' \subset E$ and a vertex $v$ we will often denote that $\delta(v) \subseteq E'$ by writing $v \in E'$. Given a graph $G = (V, E)$, two vertices $s, t \in V$, and a path $P$ from $s$ to $t$ we denote the vertices of $P$ as $V(P)$ and $E(P)$ will denote the edges of $P$. Given two vertices $u, v \in V(P)$ the path $P[u, v]$ is the subpath of $P$ which goes from $u$ to $v$. Furthermore, $P]u, v[$ will be the induced path $P[u, v] \setminus \{u, v\}$.

[8] These two types of vertices can be thought of as the rim and the hub of a wheel.

An instance of a parameterized problem $\Pi$ consists of a main part $I$ and a parameter $l$, and it is denoted denoted $(I, l)$. A parameterized decision problem $\Pi$ admits a **fixed-parameter tractable (FPT)** if for every instance $(I, \ell) \in \Pi$ the problem admits an algorithm which can decide if there is an affirmative or negative answer to the problem and the algorithm has complexity $O(f(\ell)|I|^c)$ for some positive function $f$. The problem admits a *uniform FPT algorithm*, if there is an algorithm $\mathcal{A}$ that solves an instance $(I, \ell)$ of the problem in $O(f(\ell)|I|^c)$ time. And it admits a *non-uniform FPT algorithm* if for each value of $\ell$, there is an algorithm $\mathcal{A}_\ell$ that solves an instance $(I, \ell)$ in $O(f(l)|I|^c)$ time.

### Vertex-cuts and Separators

▶ **Definition 2.** *Given a set of edges $U \subseteq E$ the **vertex-cut** induced by $U$ is the minimal set of vertices denoted $\mathcal{S}(U) \subseteq V$ such that for every pair of edges $e \in U$ and $e' \in \overline{U}$ we have $V(e) \cap V(e') \subseteq \mathcal{S}(U)$.*

Note that, for every vertex $u \in \mathcal{S}(U)$ the sets $\delta(u) \cap U$ and $\delta(u) \cap \overline{U}$ are both not empty due to minimality of $U$. For two disjoint sets of edges $U, W \subseteq E$ we will use the notation $\mathcal{D}(U, W)$ to define those vertices in $V$ which are endpoints of edges in both $U$ and $W$. For two different graphs $G = (V, E)$ and $G' = (V, E')$, defined on the same set of vertices $V$, and a set of edges $U$ such that $U \subseteq E \cap E'$ we will use a subscript to indicate in which graph we consider the vertex-cut $\mathcal{S}(U)$. That is, $\mathcal{S}(U)_G$ will refer to a (uniquely determined) vertex-cut in $G$ while $\mathcal{S}(U)_{G'}$ will refer to a (uniquely determined) vertex-cut in $G'$. Moreover, with a slight abuse of notation for the sake of brevity, for a set of edges $W \subseteq E$ and the graph $G^* = (V, E \setminus W)$ we denote the vertex-cut $\mathcal{S}(U \setminus W)_{G^*}$ in $G^*$ as $\mathcal{S}(U)_{G^*}$. Note that, the underlying graph of a vertex-cut and the corresponding subset of edges are always clear from the context. Note that $\mathcal{S}(U)_{G^*} \subseteq \mathcal{S}(U)_G$. We will omit the subscript if there is only one graph or it is explicitly given in which graph the vertex-cut should be considered. Similarly, for sets of edges and sets of vertices a subscript will indicate which graph we refer to.

A set of vertices $Q \subseteq V$ in a graph $G = (V, E)$ is a **vertex-separator** if there exist two vertices $u, v \in V \setminus Q$ such that there is no path from $u$ to $v$ in $G - Q$. We will also call such a vertex-separator $Q$ a **$(u, v)$-separator** and say that $Q$ **separates** $u$ and $v$. For a minimal $(u, v)$-separator $Q$, let $R$ be the set of vertices in the same connected component as $u$ in $G - Q$ and $T = V \setminus (R \cup Q)$. Now let $U \subseteq E$ be the set of edges which have an endpoint in $R$ and observe that $Q = \mathcal{S}(U)$, that is, the vertex-cut $\mathcal{S}(U)$ is a minimal $(u, v)$-separator. This implies that $\delta(u) \subseteq U$ and $\delta(v) \subseteq \overline{U}$. Conversely, if there exists a vertex-cut $\mathcal{S}(U)$ such that for two vertices $u, v \in V$ it holds that $\delta(u) \subseteq U$ and $\delta(v) \in \overline{U}$ then there cannot be a path from $u$ to $v$ in $G - \mathcal{S}(U)$ as $\delta(v)$ will be disconnected from $\delta(u)$ in this graph. Hence $\mathcal{S}(U)$ must be a $(u, v)$-separator. Therefore, we have the following observation:

▶ **Observation 3.** *Let $G = (V, E)$ be a graph. For $U \subseteq E$, if there exist two vertices $u, v \in V$ such that $\delta(u) \subseteq U$ and $\delta(v) \in \overline{U}$, then $\mathcal{S}(U)$ is a $(u, v)$-vertex-separator. And if $Q$ is a minimal $(u, v)$-separator, then there is a subset of edges $U \subseteq E$ such that $Q = \mathcal{S}(U)$, $\delta(u) \subseteq U$ and $\delta(v) \subseteq \overline{U}$.*

We call a vertex-cut $\mathcal{S}(U)$ a **nearby vertex-separator with respect to** $e = uv$ if exacty one of the vertices $u$ and $v$ is contained in $\mathcal{S}(U)_G$, and $\mathcal{S}(U)_{G-e}$[9] is a $(u, v)$-separator in $G - e$. Figure 1 shows a nearby vertex-separator. By Definition 2, for every $x \in \mathcal{S}(U)_{G-e}$ it holds that $\delta(x)_{G-e}$ intersects $U_{G-e}$ and $\overline{U}_{G-e}$. Hence $\mathcal{S}(U)_G = \mathcal{S}(U)_{G-e} \cup v$ for $v \in V(e)$ which implies that $|\mathcal{S}(U)_G| = |\mathcal{S}(U)_{G-e}| + 1$.

[9] Recall that, for brevity we write $\mathcal{S}(U)_{G-e}$ to denote $\mathcal{S}(U \setminus \{e\})_{G-e}$.

▶ **Observation 4.** *For a graph $G = (V, E)$ and a nearby vertex-separator $\mathcal{S}(U)_G$ with respect to $e = uv$ it holds that $|\mathcal{S}(U)_G| = |\mathcal{S}(U)_{G-e}| + 1$ and $\mathcal{S}(U)_G = \mathcal{S}(U)_{G-e} \cup v$ for some $v \in V(e)$.*

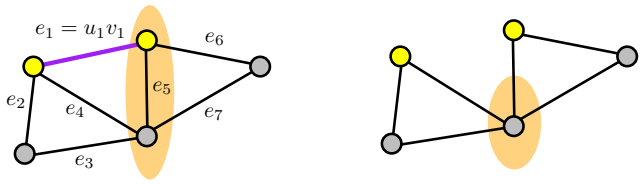



**Figure 1** $\mathcal{S}(U)$ is a near-vertex separator for $e_1$ where $U = \{e_1, e_2, e_3, e_4\}$.

▶ **Observation 5.** *A vertex-cut $\mathcal{S}(U)$ is a nearby vertex-separator with respect to $e = uv$ if and only if one of the following holds:*

- *$\delta(u) \setminus e \subseteq U$ and $\delta(v) \subseteq \overline{U}$ (or $\delta(u) \setminus e \subseteq \overline{U}$ and $\delta(v) \subseteq U$)*
- *$\delta(u) \subseteq U$ and $\delta(v) \setminus e \subseteq \overline{U}$ (or $\delta(u) \subseteq \overline{U}$ and $\delta(v) \setminus e \subseteq U$)*

We have the following results which follows directly from Menger's Theorem.

▶ **Lemma 6.** *Given a graph $G = (V, E)$ and two vertices $u, v \in V$ such that $uv \notin E$, it is possible to find a minimum $(u, v)$-separator $\mathcal{S}(U)$ in $O((|V| + |E|)^{O(1)})$ time.*

▶ **Corollary 7.** *Given a graph $G = (V, E)$ and an edge $e$, in $O((|V| + |E|)^{O(1)})$ time it is possible to find a minimum nearby vertex-separator with respect to $e$ or determine that no nearby vertex-separator with respect to $e$ exists.*

▶ **Theorem 8.** *The following statements are equivalent:*

- *The graph $G = (V, E)$ is $\kappa$-connected.*
- *$|V| \geq \kappa + 1$ and for every pair of vertices $u, v \in V$ such that $uv \notin E$ there are $\kappa$ internally disjoint paths from $u$ to $v$.*
- *$|V| \geq \kappa + 1$ and every vertex-separator has size at least $\kappa$.*

▶ **Lemma 9.** *Given a graph $G = (V, E)$, it is possible to determine if $G$ is $\kappa$-connected in $O(|G|^{O(1)})$ time .*

For vertex-cuts in general we have the following.

▶ **Lemma 10.** *The size of vertex-cuts is a submodular function, that is, given two edge-cuts $\mathcal{S}(U)$, $\mathcal{S}(W)$ we have $|\mathcal{S}(U \cap W)| + |\mathcal{S}(U \cup W)| \leq |\mathcal{S}(U)| + |\mathcal{S}(W)|$.*

### Deletable and Relevant Edges

For a p-$\kappa$-VCSS instance $(G = (V, E), \ell)$ where $G$ is a $\kappa$-vertex connected graph, we call a subset of edge $F \subseteq E$ **a solution** if $G = (V, E \setminus F)$ is $\kappa$-connected and $|F| \geq l$. To help distinguish between the edges which may be part of a solution and the edges which can definitely not be part of a solution we have the following definition:

▶ **Definition 11.** *An edge $e \in E$ is **deletable** if $G - e$ is $\kappa$-connected. If an edge is not deletable then it is **undeletable**.*

Clearly every edge of a solution $F \subseteq E$ is deletable in $G$. Let the set of deletable edges of $G$ be denoted $\mathsf{del}(\mathbf{G})$, and denote the remaining edges by $\mathsf{undel}(G)$. It means that for a solution $F \subseteq E$ we have $F \subseteq \mathsf{del}(G)$. Instead of working directly with deletable and undeletable edge we will be working with a subset with the property that $\mathcal{R} \subseteq \mathsf{del}(G)$, such that if there exists a solution $F$ then there also exists a solution $F' \subseteq \mathcal{R}$. The idea behind the algorithm is that we will either find a solution or shrink the set $\mathcal{R}$ until $\mathcal{R}$ is small enough that we can apply a brute force algorithm to determine if a solution exists or no solution exists.

▶ **Definition 12.** *A set $\mathcal{R} \subseteq \mathsf{del}(G)$ is a set of **relevant** edges if one of the following is true.*
- *There exists a solution $F \subseteq \mathcal{R}$.*
- *There exists no solution to the problem.*

Now we have the following relation between a deletable edge $e \in E$ and a nearby vertex-separators with respect to $e$.

▶ **Proposition 13.** *Given a graph $G = (V, E)$ and a deletable edge $e \in E$ it holds that every nearby vertex-separator with respect to $e$ in $G$ is of size at least $\kappa + 1$.*

## 3 Overview of the Algorithm

In this section we present an overview of our algorithm, that highlights the main ideas and techniques of this paper. Broadly we follow the approach of Bang-Jensen et al. [1], who studied the edge-connectivity version of the problem in both graphs and digraphs, and also the vertex-connectivity version in digraphs. However we develop some non-trivial new ideas and methods for vertex connectivity in undirected graphs. Due to limited space, the proofs and some technical details have been omitted from this extended abstract; they will be presented in the full version of the paper.

The starting point of our algorithm (similar to Bang-Jensen et al [1]) is the notion of a *deletable edge*. Let $\mathsf{del}(G)$ be the set of all deletable edges in $G$. Note that it is computable in polynomial time. It is clear that any solution $F$ to this instance is a subset of $\mathsf{del}(G)$. Here, by a *solution to* $(G, \ell)$ we mean a subset of $\ell$ edges, $F$, such that $G - F$ is $\kappa$-connected. Note that, if $|\mathsf{del}(G)|$ itself is upper-bounded, by say $49\kappa^2\ell^4$, then a simple brute-force algorithm can find a solution in the time $2^{\mathcal{O}(\kappa^2\ell^4)} \cdot n^{\mathcal{O}(1)}$. Therefore, we may assume that $|\mathsf{del}(G)| \geq 49\kappa^2\ell^4$.

Consider the effect of removing a deletable edge $e \in \mathsf{del}(G)$ from $G$. Observe that a number of edges in $\mathsf{del}(G)$ could become *undeletable* in $G - e$, i.e. removing any of these edges in $G - e$ will violate the $\kappa$-connectivity constraint. Let $\mathcal{I}_e^G = \mathsf{del}(G) \setminus \mathsf{del}(G - e)$ denote the set of all deletable edges in $G$ that become undeletable in $G - e$. We drop the superscript when the graph is clear from context. Now consider the following simple greedy algorithm based on the notion of deletable edges:

> If there is a deletable edge $e$ in the current instance $(G, \ell)$, find the one minimizing $|\mathcal{I}_e|$. Then recursively solve the instance $(G - e, \ell - 1)$ to obtain a solution $S'$ to it. Finally output the solution $S = S' \cup \{e\}$. Otherwise, output that there is no solution.

Observe that this algorithm succeeds only if in each sub-instance, the set $\mathcal{I}_e$ is not too large (e.g. $\mathcal{O}(\kappa^2\ell^3)$). In other words, only a bounded number of deletable edges should become undeletable in each recursive call. Then assuming that $\mathsf{del}(G)$ was sufficiently large (e.g. at least $\mathcal{O}(\kappa^2\ell^4)$) at the beginning, this greedy algorithm produces a solution to $(G, \ell)$ in polynomial time.

The remaining case is when there is a sub-instance $G'$ where, for any deletable edge $e' \in \mathsf{del}(G')$ $|\mathcal{I}_{e'}^{G'}| > 100\kappa^2\ell^3$, that is a large number of deletable edges become undeletable on removing $e'$ from $G'$. We pick such an edge $e' \in \mathsf{del}(G')$ and consider the pair $(G', e')$ and

analyze their structural properties. Our main technical result, simplified, is an algorithm $\mathcal{A}_{Irr}$ that given $G, G', e'$, where $G'$ is a $\kappa$-connected subgraph $G'$ of $G$ and $e' \in \mathsf{del}(G')$ such that $|\mathcal{I}_{e'}^{G'}| > 100\kappa^2\ell^2$, either finds a solution to $(G, \ell)$, or identifies a new irrelevant edge $e'' \in \mathcal{I}_{e'}^{G'}$ for the instance $(G, \ell)$. Here, an *irrelevant edge* refers to a deletable edge $e'' \in \mathsf{del}(G)$ such that there is a solution $F \subseteq \mathsf{del}(G) \setminus e''$. If we mark $e''$ as irrelevant, then it is treated just like an undeletable edge, and it remains in the graph.

The algorithm runs over many iterations, where in each iteration we either find a solution or find a new irrelevant edge. We maintain a set of *relevant* edges $\mathcal{R}$ which is initially $\mathsf{del}(G)$, and update it over a sequence of iterations until either a solution is found, or we have rule out the existence of a solution. Note that we ensure that $\mathcal{R}$ is always a subset of the deletable edges. Further, the definition of the set $\mathcal{I}_e$ now becomes $\mathcal{R} \setminus \mathsf{del}(G - e)$, since we are only interested in solutions formed with relevant edges. Therefore our updated algorithm for p-$\kappa$-VCSS is as follows:

> Let $(G, \ell)$ be the input instance of p-$\kappa$-VCSS, and let $\mathcal{R} \subseteq \mathsf{del}(G)$ be the set of relevant edges. We first apply the above greedy algorithm to $(G, \ell)$, and output a solution if one is found.
> Otherwise the above greedy algorithm fails to find a solution to $(G, \ell)$, therefore we find a sub-instance $G'$ and pick an arbitrary edge $e' \in \mathsf{del}(G')$ such that $\mathcal{I}_{e'}$ is large. Then, apply the algorithm $\mathcal{A}_{Irr}$ to $(G, G', e', \mathcal{R})$. It either finds a solution $F$ to $(G, \ell)$, or it finds a new irrelevant edge $e'' \in \mathcal{I}_{e'} \subseteq \mathcal{R}$ for $(G, \ell)$. In the first case, we output the solution found by $\mathcal{A}_{Irr}$. In the second case, start a new iteration on the instance $(G, \ell)$ with $\mathcal{R} - e''$ as the new set of relevant edges.

Observe that we have at most $|E|$ iterations of the above algorithm. Hence, the running time of the above algorithm is $(t_{\mathcal{A}_{Irr}} + 2) \cdot n^{\mathcal{O}(1)}$, where $t_{\mathcal{A}_{Irr}}$ denotes the running time of the algorithm $\mathcal{A}_{Irr}$. We prove that $t_{\mathcal{A}_{Irr}} = 2^{\mathcal{O}(\kappa^2\ell^4)} \cdot n^{\mathcal{O}(1)}$ where $n = |V|$.

Let us also address another special case of the problem that can be solved in polynomial time. It leads to a bound on the number of relevant edges incident on any single vertex. This is required for our implementation of $\mathcal{A}_{Irr}$. We begin by observing the following.

▶ **Observation 14.** *Given a $\kappa$-connected graph $G = (V, E)$, a set of relevant edges $\mathcal{R} \subseteq E$, and an edge $e = uv \in \mathcal{R}$. In $G - e$ every set of $\kappa$ internally disjoint paths from $u$ to $v$, $\mathcal{P} = \{P_1, P_2, \ldots, P_\kappa\}$, will use all of the irrelevant edges with respect to $e$, that is, $\mathcal{I}_e^G \subseteq \cup_{i \in \{1,\ldots,\kappa\}} E(P_i)$.*

▶ **Lemma 15.** *Given a $\kappa$-connected graph $G = (V, E)$ and a set of relevant edges $\mathcal{R}$. If there exists a vertex $u \in V$ such that $|\delta(u) \cap \mathcal{R}| \geq (\kappa + 1) \cdot l$, then a solution exists and it can be found in $O((|E| + |V|)^{O(1)})$ time.*

In the following subsections we give an overview of main ideas of the algorithm $\mathcal{A}_{Irr}$. For simplicity, we assume that $G = G'$, and we have an edge $e \in \mathcal{R} \subseteq \mathsf{del}(G)$ such that $|\mathcal{I}_e| > 100\kappa^2\ell^3$. In the general case, where $G'$ is a proper subgraph of $G$, our methods are identical with one extra step for handling the edges in $E(G) \setminus E(G')$. We construct a Wheel decomposition of the graph $G'$ and then we lift it back to $G$ (see Lemma 22).

### The Wheel Decomposition

The key conceptual idea behind the algorithm $\mathcal{A}_{Irr}$, is what we call the *Wheel decomposition* of a graph; the name is derived from the decomposition structure. See Figure 2.

▶ **Definition 16.** *Let $r \geq 3$ and $\alpha$ be integers. We say that $G$ admits an $\alpha$-wheel decomposition $(\mathcal{W}, r)$ if there is an edge partition $\mathcal{W} = \{X_0, X_1, \ldots, X_{r-1}\}$ of $E(G)$ into nonempty subsets, where the subsets are indexed cyclically, such that for every $i \in \{0, \ldots, r-1\}$ the following holds.*

1. $|\mathcal{S}(X_i)| = \alpha$
2. $\mathcal{S}(X_i) = \mathcal{D}(X_{i-1}, X_i) \cup \mathcal{D}(X_i, X_{i+1})$,
3. $|\mathcal{D}(X_{i-1}, X_i)| = |\mathcal{D}(X_i, X_{i+1})|$,
4. $\mathcal{D}(X_{i-1}, X_i) \cap \mathcal{D}(X_i, X_{i+1}) = \mathcal{S}(X_i) \cap \mathcal{S}(X_j)$ *for every* $j < i-1$ *and* $j > i+1$.

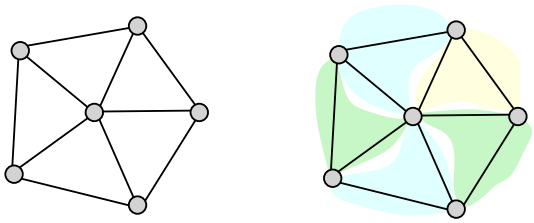


**Figure 2** A simple example of a Wheel decomposition of a graph; each part contains two edges and the center vertex is the only middle vertex of this wheel decomposition.

Observe that Property 4, ensures that for every vertex $v \in V(G)$ either $\delta(v)$ contains an edge from every part of $\mathcal{W}$, or from at most two consecutive parts of $\mathcal{W}$, where $X_{r-1}$ and $X_0$ are taken to be consecutive giving $\mathcal{W}$ a cyclic order. The vertices of the first kind, i.e. those that have an edge in every part of $\mathcal{W}$ are called the *middle vertices* of $\mathcal{W}$. We think of these vertices as the hub of a wheel whose ring is formed by the non-middle vertices of $\mathcal{W}$. Note the highly symmetrical properties of the vertex-cuts induced by the parts of $\mathcal{W}$. These are extensively applied in our proofs.

For the algorithm $\mathcal{A}_{Irr}$, we shall associate a $\kappa$-Wheel decomposition $\mathcal{W}$ with a subset of edges $E^+ \subseteq \mathcal{R}$. Recall that the input to this algorithm consists of a tuple $(G, \mathcal{R}, G', e')$, and in the simplified case that we are currently considering $G = G'$. Further, we have that $\mathcal{I}_{e'} = \mathcal{R} \setminus \mathsf{del}(G - e')$ contains at least $100\kappa^2\ell^3$ edges. We will compute a subset $E^+$ of $\mathcal{I}_e$, and use a Wheel decomposition to identify an irrelevant edge in $E^+$. Towards this we first recall the notion of a nearby vertex separator. We then have the following definition.

▶ **Definition 17.** *Let $\alpha$ be an integer. Let $G$ be a graph and let $E^+ = \{e_0, e_1, \cdots, e_{|U|-1}\} \subseteq E(G)$ be a subset of edges. We say that $G$ admits an* ***edge-restricted $\alpha$-wheel decomposition*** *$(\mathcal{W}, E^+)$ if there is an $\alpha$-wheel decomposition $(\mathcal{W}, |E^+|)$ such that for every $i \in \{0, \ldots, |E^+| - 1\}$;*

- $e_i \in X_i$,
- $\mathcal{S}(X_i)$ *and* $\mathcal{S}(X_{i+1})$ *are both nearby vertex-separators with respect to* $e_i$.

▶ **Observation 18.** *Let $G$ be a graph and let $E^+ = \{e_0, e_1, \ldots, e_{r-1}\} \subseteq \mathcal{R}$ be a subset of relevant edges. Let $(\mathcal{W}, E^+)$ be an edge-restricted $\alpha$-Wheel decomposition of $G$, for some integer $\alpha$, where $\mathcal{W} = \{X_0, X_1, \ldots, X_{r-1}\}$. Then for each edge $e_i = u_i v_i \in E^+$ such that $e_i \in X_i$, we have that $\delta(u_i) \subseteq X_i$ and $\delta(v_i) \setminus \{e_i\} \in X_{i+1}$. Here $X_r$ denotes the set $X_0$.*

The key idea is the fact that both $\mathcal{S}(X_i)$ and $\mathcal{S}(X_{i+1})$ are nearby vertex separators for $e_i$. This means that both $\mathcal{S}(X_{i+1})_{G-e_i}$ and $\mathcal{S}(X_i)_{G-e_i}$ are $u_i - v_i$ vertex separators. Hence $\delta(u_i) \setminus e_i \subseteq X_i \setminus e_i$ and $\delta(v_i) \setminus e_i \subseteq \overline{X_i}$. We can draw similar conclusions about $X_{i+1}$, and then arrive at these observations. It is immediate from Observation 18 that both endpoints of an edge $e_i \in E^+$ must be non-middle vertices of the Wheel decomposition $(\mathcal{W}, E^+)$. This

fact will be important in our proofs. However, we must also deal with another issue. Suppose that we have a $(\kappa+1)$-Wheel decomposition $(\mathcal{W}, E^+)$ and we wish to mark an edge $e_i \in E^+$ as irrelevant. To argue the correctness, we have to show that there is a solution that excludes $e_i$. Towards this, given some solution $F$ to $(G, \ell)$ that contains $e_i$, we construct another solution $F'$ that excludes $e_i$. The construction of $F'$ in our proofs must not only exclude $e_i$ but every edge in $E'_i = F \cap E(\mathcal{D}(X_i, X_{i+1})) \setminus Z$, where $Z$ denotes the set of middle vertices of the Wheel decomposition $(\mathcal{W}, E^+)$; this is a constraint of our arguments. These edges will be replaced by another set of edges contained in $E(\mathcal{D}(X_j, X_{j+1})) \setminus Z$ for some $j \neq i$. To describe the construction of $F'$ formally we require the following definition.

▶ **Definition 19.** *Given a $\kappa$-connected graph $G$, and a set of relevant edges $\mathcal{R} \subseteq E(G)$. If $G$ admits an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}, E^+)$ where $E^+ \subseteq \mathcal{R}$ then a* ***friendly set of edges for $e_i \in E^+$*** *is a set of edges $E_i \subseteq \mathcal{R}$ such that*

1. *$e_i \in E_i$,*
2. *every edge $e \in E_i$ has at least one of it's endpoints in $\mathcal{D}(X_i, X_{i+1})$ and none of the endpoints of $e$ are middle vertices,*
3. *$G - E_i$ is $\kappa$-connected,*

*and $E_i$ has the maximum cardinality among all edge subsets fulfilling these three criteria.*

Observe that a friendly set of edges $E_i$ for an edge $e_i \in E^+$ gives an upper-bound on the set $E'_i$ described above. It suggests that choosing $e_i$ that minimizes $|E_i|$ is the most likely candidate for a new irrelevant edge. The following lemma gives us a way of a computing friendly set of edges for a given wheel decomposition $(\mathcal{W}, E^+)$.

▶ **Lemma 20.** *Given a $\kappa$-connected graph $G$, a set of relevant edges $\mathcal{R}$, and an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}, E^+)$ of $G$ where $E^+ \subseteq \mathcal{R}$, it is possible to find a family of friendly sets $\mathcal{E} = \{E_i | i \in \{0, \ldots, |E^+|\}\}$ in $O(2^{O(d\cdot(\kappa+1))} \cdot |G|^{O(1)})$ time, where $d$ is the maximum number of relevant edges incident on a vertex $v \in V(G)$.*

Note that this lemma assumes that the number of relevant edges incident on any vertex is bounded by some number $d$. Here we require Lemma 15, where we have shown that if there were a vertex $v$ that had $(\kappa+1)\cdot\ell$ relevant edges incident on it, then we can compute a solution in polynomial time. Hence, we can assume that $d < (\kappa+1)\cdot\ell$, and therefore compute a friendly set of edges in $2^{\mathcal{O}((\kappa+1)^2\cdot l)} \cdot n^{\mathcal{O}(1)}$ time. This is suitable for an FPT algorithm.

With all these definitions and lemmas in hand, the process of identifying a new irrelevant edge in an instance $(G, \mathcal{R}, \ell)$ will be as follows. Using the steps described in the following sub-section, we will arrive at a subset of relevant edges $E^+ \subseteq \mathcal{R}$ containing more than $6\ell + \kappa + 2$ edges and an edge-restricted $(\kappa+1)$-Wheel decomposition $(\mathcal{W}, E^+)$ in polynomial time. Next, we will compute a friendly set of edges for every $e_i \in E^+$. Here we apply Lemma 20, along with the bound on the number of relevant edges incident to a vertex due to Lemma 15. This yields a friendly set of edges, $E_i$ for each edge in $e_i \in E^+$, in total time $2^{\mathcal{O}((\kappa+1)^2\cdot l)} \cdot n^{\mathcal{O}(1)}$. Finally, we mark the edge in $e_i \in E^+$ with the smallest friendly set of edges as irrelevant. The correctness of the last step is from the following lemma, which is one of our main technical results.

▶ **Lemma 21.** *Given a graph $G = (V, E)$, a set of relevant edges $\mathcal{R}$, an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}, E^+)$ of $G$ with $E^+ \subseteq \mathcal{R}$ where $|E^+| \geq 6l + \kappa + 2$, and a family of friendly sets $\mathcal{E}$ for the edges in $E^+$ such that $E_s \in \mathcal{E}$ has the minimum cardinality of all the sets in $\mathcal{E}$, it holds that $\mathcal{R} - e_s$ is a set of relevant edges in $G$.*

To prove this lemma we consider a hypothetical solution $F$ such that $e_s \in F$ and then we construct another solution $F'$ that excludes $e_s$. This then implies that $e_s$ is irrelevant. Let $\mathcal{W} = \{X_0, X_1, \ldots, X_{6\ell+\kappa+1}\}$. Since $|F| = \ell$, there must exist an index $i \in \{1, \ldots, 6\ell+\kappa+2\}$

such that for every $j \in \{-2, \ldots, 3\}$, $X_{i+j} \cap F = \emptyset$. Let $E'_s = F \cap \mathcal{D}(X_i, X_{i+1})$ and note that $e_s \in E'_s$. Let $F' = (F \setminus E'_s) \cup E_i$. We claim that $F'$ is also a solution. It is clear that $|F'| \geq |F|$ by our choice of $e_s$. To argue that $G - F'$ is $\kappa$-vertex connected, we give an argument based on (hypergraph) cut submodularity. While the actual argument is quite long and non-trivial, the essence of it is that if there were a $\kappa - 1$ cut in $G - F'$, then using submodularity we can trim it down to a $\kappa - 1$ cut that must exist in $G - F$. This is a contradiction since $F$ is a solution, i.e. $G - F$ is $\kappa$-connected. Here our arguments make extensive use of the highly symmetric properties of the cuts $\mathcal{S}(X_i)$.

The next lemma allows us to lift a Wheel decomposition from a subgraph $G'$ back to a graph $G$. This is required for the case where our simplifying assumption that $G' = G$ doesn't hold. In that case, $G = G' + F'$ and we have a Wheel decomposition in $G'$ that must be converted into a decomposition in $G$.

▶ **Lemma 22.** *Given a graph $G'$, a nonempty set of edges $F' \subseteq V(G') \times V(G')$, an edge-restricted $(\kappa + 1)$-wheel decomposition $(\mathcal{W}' = \{X'_0, X'_1, \cdots, X'_{|\mathcal{W}'|-1}\}, E' = \{e'_0, e'_1, \cdots, e'_{|\mathcal{W}|'-1}\})$ of $G'$ such that $|\mathcal{W}'| \geq 2 \cdot |F'| \cdot (r+1) + 1$ for an integer $r \geq 4$, it holds that in polynomial time it is possible to find an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}', E^+)$ of $G = G' + F'$ such that $|\mathcal{W}'| = r$ and $E^+ \subseteq E'$.*

The proof of this lemma essentially attempts to "merge" those parts $\{X'_i\}$ that are damaged by adding $F'$. The details are presented in the full-version.

### Wheel Decomposition for p-$\kappa$-VCSS

In this section we give an overview of the next part of the algorithm $\mathcal{A}_{Irr}$ that computes a Wheel decomposition with certain useful properties. The input is a $\kappa$-connected graph $G$ on $n$ vertices, an integer $\ell$ and a set of relevant edges $\mathcal{R}$ and an edge $e \in \mathcal{R}$ such that $\mathcal{I}_e = \mathcal{R} \setminus \mathsf{del}(G - e)$ contains at least $\kappa\ell \cdot 2(h+3)$ edges. Here $h \geq 3$ denotes the number of parts in our Wheel decomposition, and it is an integer whose value is roughly $O(\kappa\ell^2)$ in our final algorithm. We compute a edge-restricted Wheel decomposition $(\mathcal{W}, E^+)$ where $E^+ \subseteq \mathcal{I}_e$ using the following lemma, which is our second main technical result.

▶ **Lemma 23.** *Given a $\kappa$-connected graph $G = (V, E)$, an integer $h \geq 3$, an edge $e = uv \in \mathcal{R}$ where $\mathcal{R}$ is a relevant set of edges for $G$, such that $|\mathcal{I}_e| \geq \kappa \cdot \ell \cdot 2(h+3)$, in polynomial time $G$ it is possible to either:*

1. *find a set of edges $F \subseteq E$ such that $G - F$ is $\kappa$-connected and $|F| = \ell$, or*
2. *find a set $E^+ \subseteq \mathcal{I}_e$ such that $|E^+| = h$ and an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}, E^+)$ of $G$.*

Our proof of this lemma relies on several structural properties of $e$ and $\mathcal{I}_e$. In essence, we first identify a suitable subset of edges in $\mathcal{I}_e$ that can lead to the required Wheel decomposition, and then prove this fact. The first ingredient is the following. Let $e = uv$ and since $e \in \mathcal{R}$, the graph $G - e$ is $\kappa$-connected. Then we argue that for any system of $\kappa$ internally disjoint paths from $u$ to $v$ in $G - e$, $\mathcal{P}$, every edge in $\mathcal{I}_e$ occurs in one of the paths in $\mathcal{P}$. This is because if some edge $e' \in \mathcal{I}_e$ was excluded from $\mathcal{P}$, then one can show that $G - e - e'$ must also be $\kappa$-connected. From this statement we can conclude that one of the paths in $\mathcal{P}$ must have at least $|\mathcal{I}_e|/\kappa \geq \ell \cdot 2(h+3)$ edges from $\mathcal{I}_e$, and let $P \in \mathcal{P}$ be such a path. We say that this path $P$ satisfies the *intersection property* with respect to $e$. Observe that by a simple max-flow computation we can find this path $P$ in polynomial time. Let $\{e_1, e_2, \ldots, e_{\ell \cdot 2(h+3)}\} \subseteq E(P) \cap \mathcal{I}_e$ where the edges are in sequence of traversing $P$ from $u$ to $v$. Our objective is to show that a subset of these edges can lead us to the required Wheel decomposition.

Let $E' = \{e_i \mid i = 1 \mod 2(h+3)\}$, and let $r$ be the least integer such that $e_r \in E'$ and $G' = G - \{e_i \mid i = 1 \mod 2(h+3) \text{ and } i \leq r\}$ is *not* $\kappa$-connected. If $r \geq \ell + 1$, then clearly $\{e_i \mid i = 1 \mod 2(h+3) \text{ and } i < r\}$ is the required solution to $(G, \ell, \mathcal{R})$. Therefore, we assume $r \leq \ell$, and let $E'' = \{e_i \mid i = 1 \mod 2(h+3) \text{ and } i < r\}$. It is clear that $G - E''$ and $G - e_r$ are $\kappa$-connected, but $G' = G - E'' - e_r$ is not.

Let us fix a vertex-separator $\mathcal{S}(U')_{G'}$ of size less than $\kappa$ in $G'$, where $U' \subseteq E(G')$. It is clear that $\mathcal{S}(U')_{G'}$ separates the vertex pairs $u_r, v_r$ and $u_j, v_j$ where $e_r = u_r v_r$ and $e_j = u_j v_j \in E''$ for some $j < r$. Let us fix the largest such $j$. Note that $r - j \geq 2(h+3)$ and hence $r \geq 2(h+3)$. Let $E^* = \{e_j, e_r\} \cup \{e_{r+2i-2(h+2)} \mid 0 \leq i \leq h+1\}$. Finally, we define the edge subset $E^+$ which contains $h$ edges, that forms one half of our Wheel decomposition $(\mathcal{W}, E^+)$ as follows:

$$E^+ = E^* \setminus \{e_j, e_{r-2(h+2)}, e_{r-2}, e_r\}$$

We remark that the choice of $E^+$ may seem rather arbitrary, however it was chosen to encode several useful properties that are required for our proof. These properties are applied in the construction of the edge partition $\mathcal{W}$, which forms the other half of our decomposition.

For convenience, let us rename the edges in $E^*$ as $s_i \leftarrow e_{r+2i-2(h+2)}$ and further let $s = e_j$ and $t = e_r$. The following lemma is our next key ingredient, that computes a family of vertex cuts, defined via edge-subsets, corresponding to $E^*$. These cuts will be used to define $\mathcal{W}$ .

▶ **Lemma 24.** *Given a graph $G = (V, E)$, an edge $e = uv \in \mathcal{R}$ and a path $P$ between $u$ and $v$ fulfilling the intersection property such that $\mathcal{I}_e \cap P \neq \emptyset$, in polynomial time it is possible to find a family of cuts $\mathcal{C} = \{C_1, C_3 \cdots, C_{|\mathcal{I}_e \cap P|}\}$ such that for each $e_i \in \mathcal{I}_e \cap E(P)$,*

1. *$e_i \in C_i$, $e \in \overline{C_i}$, and $\mathcal{S}(C_i)$ is a nearby vertex-separator with respect to both $e_i$ and $e$.*
2. *$v_i$ is the only internal vertex of $P$ in $\mathcal{S}(C_i)$.*
3. *$|\mathcal{S}(C_i)| = \kappa + 1$,*
4. *For every $e_j = u_j v_j$ where $1 < j < i$, $\delta(u_j), \delta(v_j) \subseteq C_i$*
5. *For every $e_j = u_j v_j$ where $j \geq i + 2$, $\delta(u_j), \delta(v_j) \subseteq \overline{C_i}$*

*Further $C_i \subset C_j$ for any $j \geq i + 2$.*

Proving this lemma is not too difficult, although it requires working out some details. The idea is to start with a collection of arbitrary $\kappa$-cuts in $G - e$ containing an endpoint of these edges, and further exploiting the membership of $e_i$ in $\mathcal{I}_e$. These cuts are "untangled" using the submodularity of hypergraph cuts, to obtain that last containment property. Similar ideas were used by Bang-Jensen et al [1].

Applying the above lemma to $G, e$ and $P$ we obtain a collection of cuts, $\mathcal{C} = \{C_r, C_0, C_1, \ldots C_{h+1}, C_t\}$ corresponding to the edges in $E^* = \{s, s_0, s_1, \ldots, s_{h+1}, t\}$. Note that these cuts satisfy all the properties of the above lemma, and in particular $C_i \subset C_j$ for any $i < j$ in this collection. We can now define the edge-partition of our Wheel decomposition as follows:

$$\mathcal{W} = \{X_1 = E \setminus (C_h \cap \overline{C_1} \cap U'\} \cup \{X_i = C_i \cap \overline{C_{i-1}} \cap U' \mid 2 \leq i \leq h\}$$

Finally, to prove that $\mathcal{W}$ is indeed the required Wheel decomposition, we prove it's properties one by one. The main tool that we use is once again submodularity of cuts. In particular, from the definition of $\mathcal{W}$, some intutition can be derived. The cut induced by $U'$ is of size $\kappa - 1$ in $G'$. This cut splits each $C_i$ into two pieces, and further because of the containment property of $\mathcal{C}$, the parts $X_i$ are defined as the portion of $C_i \setminus C_{i-1}$ in $U'$. The part $X_1$ simply contains the remaining edges. The various symmetric properties of the cuts induced by each $X_i$ are consequences of the choice of $E^*$, via submodularity of cuts.

### Putting it all together: An algorithm for p-$\kappa$-VCSS

We collect all the previous results to arrive at the following algorithm for p-$\kappa$-VCSS:

1. If the induced graph $G' = (V, \mathcal{R})$ contains a vertex with degree at least $(\kappa+1)\cdot l$ then use the algorithm of Lemma 15 to find a solution and return it.
2. If $|\mathcal{R}| \leq 49\kappa^2 l^4$, then for every subset $F' \subseteq \mathcal{R}$ of size $l$ determine if $F'$ is a solution. If $F'$ is a solution, return $F'$ as a solution. If none of the subsets F' is a solution, return that no solution exists.
3. Choose an edge $f_i \in \mathcal{R}_i$, and let $\mathcal{I}_{f_i} = \mathcal{R}_i \setminus \mathsf{del}(G_i \setminus f_i)$:
   a. If $|\mathcal{I}_{f_i}| \geq \kappa \cdot l \cdot 2(2i\cdot(6l+\kappa+3)+4)$ then by Lemma 23
      i. we can either find a solution $F'$ for $G_i$ of size $l$, and return $F'$ as a solution,
      ii. or we can find a set $E^+ \subseteq \mathcal{I}_e$ such that $|E^+| = 2i\cdot(6l+\kappa+3)+1$ and an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}, E^+)$ for $G_i$.
         A. If $F \neq \emptyset$ then given $G_i$, $F$, and $(\mathcal{W}, E^+)$ by Lemma 22 find an edge-restricted $(\kappa+1)$-wheel decomposition $(\mathcal{W}', E')$ of $G = G_i + F$ such that $E' \subseteq E^+$ and $|E'| \geq 6l+\kappa+2$.
         B. Otherwise, if $F = \emptyset$ then set $(\mathcal{W}', E') = (\mathcal{W}, E^+)$ to be an edge-restricted $(\kappa+1)$-wheel decomposition of $G = G_i$.
         C. Given the set of relevant edges $\mathcal{R}$, and $(\mathcal{W}', E')$ use the algorithm from Lemma 20 to find the friendly sets for every edge $e_j \in E'$.
         D. Find the edge $e_j \in E'$ for which the corresponding friendly set $E_j$ has the smallest cardinality among all the friendly sets. Now set $\mathcal{R} := \mathcal{R} \setminus e_j$ and go to step 2. The correctness of this step follows from Lemma 21.
   b. Otherwise, set $G_{i+1} = G_i - f_i$, $\mathcal{R}_{i+1} = \mathcal{R} \cap \mathsf{del}(G_{i+1})$, and $F := F \cup f_i$. If $|F| \geq l$ return $F$ as a solution.

This algorithm proves Theorem 1. The correctness follows from the correctness of each lemma used in the above. For the running time, observe that Step 1 runs in polynomial time. The next steps are repeated at most $|E(G)|$ times, for each iteration, where we shrink the set of relevant edges $\mathcal{R}$ until Step 2 is finally applicable. Note that Step 2 runs in time $2^{\mathcal{O}(\kappa^2 \ell^4)} \cdot n^{\mathcal{O}(1)}$. One of Steps 3(a)i and 3(a)ii is run in each iteration, which is decided in polynomial time by Lemma 23. In the second case, we obtain a Wheel decomposition $(\mathcal{W}, E^+)$ in Step 3(a)iiC we spend $2^{\mathcal{O}((\kappa+1)^2\cdot l)} \cdot n^{\mathcal{O}(1)}$ time computing friendly sets of edges for $E^+$. We then mark an irrelevant step, and proceed to the next iteration. A straightforward calculation gives the claimed running time of the algorithm.

Due to space constraints, several proofs were omitted. Complete details may be found in the full version of the paper, in the appendix. Finally, we also thank anonymous reviewers for their suggestions on improving this paper.

## References

1 Jørgen Bang-Jensen, Manu Basavaraju, Kristine Vitting Klinkby, Pranabendu Misra, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms for survivable network design with uniform demands. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2838–2850. SIAM, 2018.

2 Jørgen Bang-Jensen and Anders Yeo. The minimum spanning strong subdigraph problem is fixed parameter tractable. *Discrete Applied Mathematics*, 156(15):2924–2929, 2008.

3 Jørgen Bang-Jensen and Gregory Gutin. *Digraphs. Theory, Algorithms and Applications*. Springer, 2002.

4 Manu Basavaraju, Fedor V Fomin, Petr Golovach, Pranabendu Misra, MS Ramanujan, and Saket Saurabh. Parameterized algorithms to preserve connectivity. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 800–811. Springer, 2014.

5 Jaroslaw Byrka, Fabrizio Grandoni, and Afrouz Jabal Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 815–825. ACM, 2020. doi:10.1145/3357713.3384301.

6 Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 167–176, 2008.

7 Joseph Cheriyan and László A Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Computing*, 43(4):1342–1362, 2014.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Science & Business Media, 2015.

9 Andreas Emil Feldmann, Anish Mukherjee, and Erik Jan van Leeuwen. The parameterized complexity of the survivable network design problem. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 37–56. SIAM, 2022.

10 Fedor V. Fomin, Petr A. Golovach, William Lochet, Pranabendu Misra, Saket Saurabh, and Roohani Sharma. Parameterized complexity of directed spanner problems. In Yixin Cao and Marcin Pilipczuk, editors, *15th International Symposium on Parameterized and Exact Computation, IPEC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference)*, volume 180 of *LIPIcs*, pages 12:1–12:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.IPEC.2020.12.

11 Greg N Frederickson and Joseph Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, 1981.

12 Gregory Gutin, M.S. Ramanujan, Felix Reidl, and Magnus Wahlström. Path-contractions, edge deletions and connectivity preservation. *Journal of Computer and System Sciences*, 101:1–20, 2019. doi:10.1016/j.jcss.2018.10.001.

13 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

14 Samir Khuller. Approximation algorithms for finding highly connected subgraphs. In *Approximation algorithms for NP-hard problems*, pages 236–265. PWS Publishing Co., 1996.

15 Kristine Vitting Klinkby, Pranabendu Misra, and Saket Saurabh. Strong connectivity augmentation is FPT. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, pages 219–234. SIAM, 2021. doi:10.1137/1.9781611976465.15.

16 Yusuke Kobayashi. Np-hardness and fixed-parameter tractability of the minimum spanner problem. *Theor. Comput. Sci.*, 746:88–97, 2018. doi:10.1016/j.tcs.2018.06.031.

17 Yusuke Kobayashi. An FPT algorithm for minimum additive spanner problem. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.11.

18 Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004. doi:10.1137/S0097539702416736.

19 Guy Kortsarz and Zeev Nutov. Approximating minimum cost connectivity problems. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2010.

20 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.135.

21 Dániel Marx and László A Végh. Fixed-parameter algorithms for minimum-cost edge-connectivity augmentation. *ACM Transactions on Algorithms (TALG)*, 11(4):27, 2015.

22 Zeev Nutov. Approximability status of survivable network problems, 2014.

23 Zeev Nutov. Approximating minimum-cost edge-covers of crossing biset-families. *Combinatorica*, 34(1):95–114, 2014.

24 Vera Traub and Rico Zenklusen. A better-than-2 approximation for weighted tree augmentation. *CoRR*, abs/2104.07114, 2021. arXiv:2104.07114.

25 Vera Traub and Rico Zenklusen. A $(1.5+\epsilon)$-approximation algorithm for weighted connectivity augmentation. *arXiv preprint arXiv:2209.07860*, 2022.

26 László A Végh. Augmenting undirected node-connectivity by one. *SIAM Journal on Discrete Mathematics*, 25(2):695–718, 2011.

# Lyndon Arrays in Sublinear Time

**Hideo Bannai**
M&D Data Science Center, Tokyo Medical and Dental University, Japan

**Jonas Ellert**
Technical University of Dortmund, Germany

## Abstract

A Lyndon word is a string that is lexicographically smaller than all of its non-trivial suffixes. For example, `airbus` is a Lyndon word, but `amtrak` is not a Lyndon word due to its suffix `ak`. The Lyndon array stores the length of the longest Lyndon prefix of each suffix of a string. For a length-$n$ string over a general ordered alphabet, the array can be computed in $\mathcal{O}(n)$ time (Bille et al., ICALP 2020). However, on a word-RAM of word-width $w \geq \log_2 n$, linear time is not optimal if the string is over integer alphabet $\{0, \ldots, \sigma\}$ with $\sigma \ll n$. In this case, the string can be stored in $\mathcal{O}(n \log \sigma)$ bits (or $\mathcal{O}(n/\log_\sigma n)$ words) of memory, and reading it takes only $\mathcal{O}(n/\log_\sigma n)$ time. We show that $\mathcal{O}(n/\log_\sigma n)$ time and words of space suffice to compute the succinct $2n$-bit version of the Lyndon array. The time is optimal for $w = \mathcal{O}(\log n)$. The algorithm uses precomputed lookup tables to perform significant parts of the computation in constant time. This is possible due to properties of periodic substrings, which we carefully analyze to achieve the desired result. We envision that the algorithm has applications in the computation of runs (maximal periodic substrings), where the Lyndon array plays a central role in both theoretically and practically fast algorithms.

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

**Keywords and phrases** Lyndon forest, Lyndon table, Lyndon array, sublinear time algorithms, word RAM algorithms, word packing, tabulation, lookup tables, periodicity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.14

**Funding** *Hideo Bannai*: JSPS KAKENHI Grant Number JP20H04141

## 1 Introduction

A Lyndon word is a string that is lexicographically smaller than all of its non-trivial suffixes. For example, `airbus` is a Lyndon word, but `amtrak` is not a Lyndon word due to its suffix `ak`. The Lyndon array stores the length of the longest Lyndon prefix of each suffix of a string (a precise definition follows later). In this article, we propose a new algorithm that computes the (succinct version of) the Lyndon array of a length-$n$ string over alphabet $\{0, \ldots, \sigma\}$ in $\mathcal{O}(n/\log_\sigma n)$ time and words of space on a word-RAM of word-width $w \geq \log_2 n$.

**Background and Applications.** Since their introduction in the field of combinatorics on words almost 70 years ago [32], Lyndon words have proven to be useful for designing efficient algorithms. The Lyndon factorization of a string uniquely decomposes it into lexicographically non-increasing Lyndon words [9, 15]. It can easily be obtained from the Lyndon array, and it has recently been used to capture overlaps between reads for next generation DNA sequencing [7, 8]. The closely related standard factorization $T = RS$ of a Lyndon word $T$ is uniquely defined by $S$, which is the longest proper Lyndon suffix, or equivalently the lexicographically smallest proper suffix of $T$. It is guaranteed that $R$ is also a Lyndon word [9]. By recursively factorizing $R$ and $S$ in the same manner until all segments are single symbols, we obtain the binary Lyndon tree. This tree can also be defined for a non-Lyndon word $T$, since prepending an infinitely small symbol makes any word Lyndon. The Lyndon tree encodes the same information as the Lyndon array (see, e.g., [13, 2]).


Hohlweg and Reutenauer [24] showed that the Lyndon factorization encodes the list of left-to-right suffix (lexicographical) minima of a string. Crochemore and Russo [13] showed that the Lyndon array (respectively the Lyndon tree) of a string encodes the left-to-right minima tree (respectively the Cartesian tree) of its inverse suffix array (the array that stores for each suffix its lexicographical rank among all the suffixes). This shows the close relation between the Lyndon array and the suffix array [33], one of the major data structures in string algorithmics, with countless applications in compression and indexing. Lyndon words and the Lyndon array have been used to efficiently compute the suffix array (e.g., [3, 5, 38]) and vice versa (e.g., [21, 31]). The Lyndon array can be computed from the suffix array in $\mathcal{O}(n)$ time; however, this requires a linearly-sortable alphabet (e.g., $\Sigma = \{0, \ldots, n^{\mathcal{O}(1)}\}$) due to the information-theoretic lower bound on comparison sorting. For general ordered alphabets, there is an $\mathcal{O}(n)$ time algorithm that computes the Lyndon array without the suffix array by exploiting combinatorial properties of Lyndon words [6] (see [16] for a simplified description). This algorithm can also be used to output a succinct $2n$-bit encoding of the Lyndon array, which is based on a balanced parentheses sequence of the tree structure of the Lyndon array.

The perhaps most important application of the Lyndon array is the computation of runs (maximal periodic substrings). Kolpakov and Kucherov showed that there are $\mathcal{O}(n)$ runs in a length-$n$ string, and conjectured that this upper bound can be improved to $n$ [29]. A series of results gradually improved the best known bound [40, 10, 11, 39]. Ultimately, the Lyndon array and its rich combinatorial properties were used to prove the conjecture [4], which also resulted in a remarkably simple proof. The Lyndon array is also one of the two main ingredients of a simple $\mathcal{O}(n)$ time algorithm for computing all the runs (see, e.g., [4, 13]). The second ingredient is a data structure for longest common extensions (LCEs), which answers queries of the type "Given $i, j$, what is the longest shared prefix between $T[i..n]$ and $T[j..n]$?". For linearly-sortable alphabets, an LCE data structure with constant query time can be constructed in $\mathcal{O}(n)$ time (e.g., [20]), which results in an $\mathcal{O}(n)$ time runs algorithm. It was conjectured that the same time can be achieved for general ordered alphabets [30], which resulted in a series of new LCE data structures aimed at these alphabets [30, 22, 12]. The first algorithm that achieves linear time [17] and hence proves the conjecture does not use an LCE data structure at all. Instead, it relies on the combinatorial structure of the Lyndon array to explicitly compute all the LCEs in overall linear time.

The existing $\mathcal{O}(n)$ time algorithms for the Lyndon array are optimal if the string is over a general ordered alphabet. However, they are not optimal on a word-RAM of word-width $w \geq \log_2 n$ if the alphabet is $\{0, \ldots, \sigma\}$ with $\sigma \ll n$. In this case, $\lfloor \log_\sigma n \rfloor$ symbols can be packed in a single word of memory, and hence they can be processed simultaneously. Word packing has lead to faster algorithms for many problems in string algorithmics. For example, $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time suffices to construct any of the following data structures for a packed string: compressed suffix arrays and trees with $\mathcal{O}(\log^\epsilon n)$ time operations [27]; an index that counts occurrences of a length-$m$ query pattern in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time [27]; the Burrows–Wheeler transform [26]; the wavelet tree [1, 35] with a fast practical implementation [25]. A novel LCE data structure by Kempa and Kociumaka can be constructed in even faster $\mathcal{O}(n/\log_\sigma n)$ time and answers queries in $\mathcal{O}(1)$ time [26].

**Our Contributions.** We show that the succinct $2n$ bit representation of the Lyndon array can be computed in $\mathcal{O}(n/\log_\sigma n)$ time and words of space on a word RAM of word-width $w \geq \log_2 n$. The time bound is optimal under the common assumption that the input size scales with the word-width, i.e., $w = \mathcal{O}(\log n)$. The algorithm uses the same ideas as previous $\mathcal{O}(n)$ time algorithms, but processes the string one word (rather than one position) at a time.

We accelerate the computation with lookup tables and the LCE data structure by Kempa and Kociumaka [26]. By carefully analyzing properties of periodic substrings, we are able to design the lookup tables in a way that minimizes the number of required LCE queries. The new algorithm will hopefully lead to a sublinear time algorithm for the computation of runs.

The remainder of the paper is structured as follows. In Section 2, we introduce basic definitions and notation, as well as a simple linear time algorithm for the succinct Lyndon array. This algorithm is the starting point of our new sublinear time algorithm, which we present in Section 3. It uses some auxiliary data structures, which we first use as a black box. In Sections 4 and 5, we show how to implement these data structures.

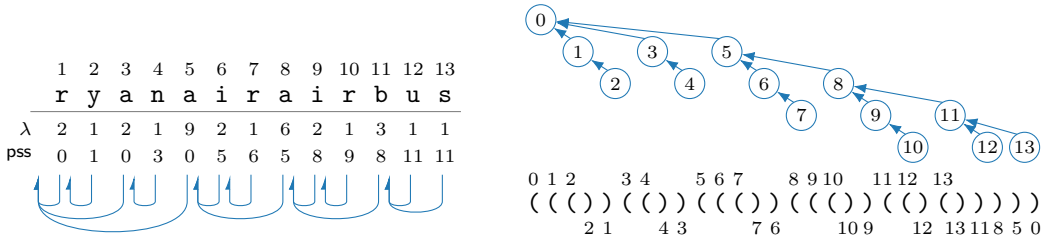
## 2 Preliminaries

**Strings and Computational Model.** For $i, j \in \mathbb{N}$, we write $[i, j] = (i-1, j] = [i, j+1) = (i-1, j+1)$ to denote the integer interval $\{i, i+1, \ldots, j\}$ (or the empty set if $i > j$). A string $T \in \Sigma^n$ is a sequence of $|T| = n$ symbols from some alphabet $\Sigma$. The empty string of length 0 is denoted by $\varepsilon$. For $i, j \in [1, n]$, we denote the $i^{\text{th}}$ symbol of the sequence by $T[i]$. The substring $T[i..j] = T(i-1..j] = T[i..j+1) = T(i-1..j+1)$ is the sequence of length $j-i+1$ that starts with the $i^{\text{th}}$ and ends with the $j^{\text{th}}$ symbol. For $j < i$ we define $T[i..j] = \varepsilon$. If $i \leq j$, then the longest common extension (LCE) at positions $i$ and $j$ is defined as $\text{LCE}(i, j) = \text{LCE}(j, i) = \max(\{\ell \in [0, n-j+1] \mid T[i..i+\ell) = T[j..j+\ell)\})$. Substrings $T[1..i]$ and $T[i..n]$ are respectively called prefix and suffix of $T$. A substring $T[i..j]$ is proper if $T[i..j] \neq T$, and non-trivial if it is proper and $T[i..j] \neq \varepsilon$. If the alphabet $\Sigma$ is totally ordered, then it induces a lexicographical order as follows. For strings $S \in \Sigma^m$ and $T \in \Sigma^n$, it holds $S \prec T$ (and we say that $S$ is lexicographically smaller than $T$) if and only if either $S$ is a prefix of $T$, or there is some $\ell \in [1, \min(m, n)]$ such that $S[1..\ell) = T[1..\ell)$ and $S[\ell] < T[\ell]$. We write $S \preceq T$ to denote that $T$ is not lexicographically smaller than $S$. A string $T[1..n]$ has period $p \in [1, n]$ if $T[1..n-p] = T[1+p..n]$. We then call $T[1..n-p]$ a border of $T$. The concatenation of two string $S$ and $T$ is denoted by $ST$. For $k \in \mathbb{N}^0$, the $k$-times concatenation (or $k$-power) of $T$ is denoted by $T^k$ (with $T^0 = \varepsilon$).

We work on a word RAM (see, e.g., [23]) with words of width $w \geq \log_2 n$ bits (where $n$ is the length of the input string). A string $T \in [0, \sigma)^n$ can be stored in packed representation, i.e., the binary representation of each symbol is stored in $\lceil \log_2 \sigma \rceil$ bits, and the entire string occupies $n \lceil \log_2 \sigma \rceil$ consecutive bits or $\mathcal{O}(n/\log_\sigma n)$ words of memory. (The number of bits can be improved to $\lceil n \log_2 \sigma \rceil + \mathcal{O}(\log^2 n)$ while retaining fast access [14, Theorem 1].) Primitive bitwise operations suffice to extract any substring of length $\mathcal{O}(\log_\sigma n)$ in constant time because such a substring fits in a constant number of words. Since a (sub-)string $S$ is a bit string of length $|S| \cdot \lceil \log_2 \sigma \rceil$, it can be interpreted as an integer in range $[1, 2^{|S| \cdot \lceil \log_2 \sigma \rceil}]$ (by reading the bit string as a binary number and adding one). We write $\mathsf{int}(S)$ to denote the integer value associated with $S$ (we also use this notations for bitvectors, as they are merely strings with $\sigma = 2$). In this model of computation, a data structure by Kempa and Kociumaka can be constructed in sublinear time and answers LCE queries (i.e., outputs the LCE of any two positions) in constant time.

▶ **Lemma 1** ([26, Theorem 5.4]). *Given a string $T \in [0, \sigma)^n$ in packed representation, LCE queries can be answered in $\mathcal{O}(1)$ time after an $\mathcal{O}(n/\log_\sigma n)$ time preprocessing.*

We do not use uninitialized memory or similar techniques, and hence the words of space used by the algorithm is upper bounded by the time spent (this includes Lemma 1). Therefore, we will not discuss any space complexities in the remainder of the paper, and instead only show the $\mathcal{O}(n/\log_\sigma n)$ time bound, which implies that $\mathcal{O}(n/\log_\sigma n)$ words of space suffice.




**(a)** Lyndon array $\lambda$ and PSS array pss. **(b)** The previous smaller suffix tree and its BPS.

**Figure 1** Data structures for the string `ryanairairbus`. Edges point to previous smaller suffixes.

**Lyndon Arrays.** A Lyndon word is a non-empty string $T[1..n]$ over a totally ordered alphabet that is lexicographically smaller than its non-trivial suffixes, i.e., $\forall i \in [2, n] : T \prec T[i..n]$.

▶ **Definition 2.** *Let $T[1..n]$ be a string over a totally ordered alphabet. The Lyndon array $\lambda[1..n]$ and the previous smaller suffix (PSS) array* pss$[1..n]$ *are defined $\forall i \in [1, n]$ by*

- $\lambda[i] = \max(\{\ell \in [1, n - i + 1] \mid T[i..i + \ell)$ *is a Lyndon word*$\})$, *and*
- pss$[i] = \max(\{j \in [1, i) \mid T[j..n] \prec T[i..n]\} \cup \{0\})$.

An example is provided in Figure 1a. We may interpret the PSS array as a rooted tree. The root is an artificial node with label 0. Every text position is a node, and for any position $i$, there is an edge from $i$ to its parent pss$[i]$. Each position is a child of a smaller position or the artificial root node, and hence it is easy to see that this indeed yields a tree with root 0. An example of this so-called *previous smaller suffix tree (PSS tree)* is provided in Figure 1b.

A balanced parentheses sequence (BPS) encodes the tree in $2n + 2$ bits (see, e.g., [36]), which can be described in the following constructive way. We write the sequence in an append-only manner. We perform a depth-first traversal of the tree, during which we visit the children of each node in increasing order. Whenever we walk down an edge from pss$[i]$ to $i$, we append the opening parenthesis of node $i$. Whenever we walk up an edge from $i$ to pss$[i]$, we write the closing parenthesis of node $i$. An example is provided in Figure 1b. If we assign preorder-numbers during this traversal, then node $i$ has preorder-number $i$ (see [19, Lemma 1] and [6]). Hence the $i^{\text{th}}$ opening parenthesis belongs to node $i$. In practice, the parentheses sequence is a bitvector, where 1-bits are opening parentheses, and 0-bits are closing ones. Bille et al. [6] showed that the PSS tree inherently encodes the Lyndon array because $\lambda[i]$ is exactly the size of the subtree rooted in node $i$. Hence we need to augment the parentheses sequence such that subtree-size$(i)$ can be answered in constant time. There are multiple support data structures that achieve this with $o(n)$ bits of additional space (see, e.g., [37]), but their current construction algorithms do not achieve sublinear time. However, we only require a small and simple subset of operations supported by these data structures (namely rank, select, and find-close). We plan to cover the efficient construction of support data structures for these operations in a future full version of the paper.

**Simple Construction Algorithm for the PSS Tree.** A simple algorithm computes the PSS tree in $\mathcal{O}(n)$ time, which will be the starting point for the $\mathcal{O}(n/\log_\sigma n)$ time algorithm in Section 3. Suppose that we have already computed the subtree induced by nodes $[0, i)$. Attaching node $i$ requires finding pss$[i]$. A strategy for this follows from the property that

**(a)** pss$[i]$ is one of the nodes on the already computed path from $i - 1$ to the root 0, and
**(b)** on this path, pss$[i]$ is the deepest node $j$ such that either $j = 0$ or $T[j..n] \prec T[i..n]$.

This observation has been used by previous results, and it has been proven, e.g., in [16, Lemma 6]. It implies an algorithm in which the positions are inserted into the tree one by one in left-to-right order. This is also the approach of Algorithm 1, which directly computes the BPS of the tree (see [6, Algorithm 1] for a more abstract version of this algorithm).

**Algorithm 1** Linear time construction of the PSS tree.

**Require:** Packed string $T \in [0, \sigma)^n$.
**Ensure:** BPS $\mathcal{B}$ of the PSS tree of $T$.

```
1:  B ← (                                   ▷ opening parenthesis of node 0
2:  Q ← stack that contains only 0
3:  for i = 1 to n do
4:      j ← Q.top()
5:      while j > 0 and T[i..n] ≺ T[j..n] do   ▷ evaluate with LCE data structure
6:          append ) to B                   ▷ closing parenthesis of node j
7:          Q.pop()
8:          j ← Q.top()
9:      append ( to B                       ▷ opening parenthesis of node i
10:     Q.push(i)
11: append |Q| times ) to B                 ▷ closing parentheses of nodes on path from n to 0
```

At the time at which the algorithm starts processing position $i$, the sequence $\mathcal{B}$ contains the prefix of the BPS that ends with the opening parenthesis of node $i-1$, and the stack $\mathcal{Q}$ contains exactly the nodes on the path from $i-1$ (topmost stack element) to the root 0 (bottommost stack element). A loop is used to find the topmost element $j$ on the stack that satisfies $j = 0$ or $T[j..n] \prec T[i..n]$ (lines 4–8). By properties (a) and (b), the final value of $j$ is the previous smaller suffix of $i$, which means that node $i$ will be attached as a child of $j$. Hence we pop the nodes on the path from $i-1$ to $j$ (but excluding $j$) from the stack, and then push $i$ on the stack (lines 7 and 10). As explained earlier, the BPS encodes a depth-first traversal of this tree. In terms of this traversal, we just moved from node $i-1$ up to node $j$, and then down to node $i$. Thus, we write one closing parenthesis for each step up (line 6), and then one opening parenthesis for moving down to node $i$ (line 9). After processing position $n$, we write the closing parentheses of the nodes on the path from $n$ to 0 (line 11).

The correctness follows from properties (a) and (b). Each line takes constant time, except for the lexicographical suffix comparison in line 5. It holds $T[i..n] \prec T[j..n]$ if and only if either $\text{LCE}(i,j) = n-i+1$ or $T[i+\text{LCE}(i,j)] < T[j+\text{LCE}(i,j)]$. Thus, the LCE data structure from Lemma 1 suffices to lexicographically compare suffixes in constant time (we use this technique repeatedly throughout the paper). The number of inner loop iterations is less than the number of closing parentheses, and hence the total time needed by the algorithm is $\mathcal{O}(n)$.

## 3 A Blockwise Algorithm for the PSS Tree

In this section, we modify Algorithm 1 such that instead of processing a single index at a time, it processes blocks of indices in each step. The block size $k = \left\lfloor \frac{\log_2 n}{8\lceil \log_2 \sigma \rceil} \right\rfloor$ is approximately one eighth of the number of symbols that fit into one word of memory, and hence there are $N = \lceil \frac{n}{k} \rceil = \Theta(n/\log_\sigma n)$ blocks. Let $B_1, \ldots, B_N$ with $\forall x \in [1, N] : B_x = (xk - k, xk]$ be the sequence of blocks (where without loss of generality we assume that $k$ divides $n$).

In the PSS tree, each block $B_x$ induces a forest that contains exactly the nodes that are members of the block. For any node $j \in B_x$, if $\mathsf{pss}[j] \in B_x$, then $\mathsf{pss}[j]$ is the parent of $j$ in the forest induced by $B_x$. Otherwise, $j$ is the root of a tree in the forest. We call these trees *small trees*, and their roots *small roots*. The small roots are exactly the left-to-right lexicographical minima of suffixes starting in $B_x$, i.e., $i \in B_x$ is a root if and only if $\forall i' \in B_x : i' < i \implies T[i'..n] \succ T[i..n]$. Just like in the PSS tree, we arrange the children of each node in increasing order. The BPS of the forest is the concatenation of the BPSs of its small trees in left-to-right order.

**High-Level Description of the Blockwise Algorithm.** We process the blocks one at a time in left-to-right order. At the time at which we process block $B_x$, we have already computed the partial PSS tree induced by all previous blocks, i.e., by $[0, xk - k]$. For $B_x$, we first obtain its induced PSS forest. Our goal is to attach the small roots (including their small trees) to the respective previous smaller suffixes, which lie on the path from $xk - k$ to 0 in the partial PSS tree. This is schematically shown in Figure 2a. Note that small roots further to the right will be attached further up in the path. This is because suffixes on the path are lexicographically decreasing towards the root, while the suffixes corresponding to small roots are lexicographically decreasing from left to right. Hence our task is to lexicographically *interleave* the path with the small roots. For an efficient implementation of this interleaving process, it is crucial that we maintain the path from $xk - k$ to 0 in a blockwise manner. Just like in Algorithm 1, we maintain a stack of the nodes on the path. However, each stack element is a pair $(y, \mathcal{L})$, where $y$ indicates that we consider block $B_y$, and $\mathcal{L}[1..k]$ is a bitvector indicating which of the positions in the block are relevant. For $j' \in [1, k]$, it holds $\mathcal{L}[j'] = 1$ if and only if $yk - k + j'$ lies on the path from $xk - k$ to 0. The stack then contains exactly the blocks with at least one 1-bit in the bitvector. This is visualized in Figure 2a.

## 3.1 Detailed Description of the Blockwise Algorithm

So far, we described the algorithm in terms of the PSS tree. However, we want to directly compute its BPS. After $\mathcal{O}(N)$ preprocessing time, we can obtain the BPS of the forest induced by any block in $\mathcal{O}(1)$ time. This follows directly from the lemma below.

▶ **Lemma 3.** *Let $T \in [0, \sigma)^n$ be a string in packed representation and let $\epsilon \in \mathbb{R}^+$. After $\mathcal{O}(n/\log_\sigma n)$ preprocessing time, the following type of query can be answered in $\mathcal{O}(1)$ time. Given a range $[i, i + \ell) \subseteq [1, n]$ of length $\ell \leq \frac{\log_2 n}{(2+\epsilon)\lceil \log_2 \sigma \rceil}$, output the BPS of the PSS forest induced by $[i, i + \ell)$, as well as a bitvector $\mathcal{R}[1..\ell]$ such that for $j \in [1, \ell]$ it holds $\mathcal{R}[j] = 1$ if and only if $i + j - 1$ is the root of a tree in the forest.*

The proof of the lemma is provided in Section 4. When we lexicographically interleave the suffixes, we will repeatedly encounter another type of query. Given a small root, we have to find its previous smaller suffix within a block on the stack. A solution for this is provided by the lemma below, which we prove in Section 5.

▶ **Lemma 4.** *Let $T \in [0, \sigma)^n$ be a string in packed representation and let $\epsilon \in \mathbb{R}^+$. After $\mathcal{O}(n/\log_\sigma n)$ preprocessing time, we can answer the following type of query in $\mathcal{O}(1)$ time. Given a position $i \in [1, n]$ and a non-empty interval $[j, j+\ell) \subseteq [1, n]$ of length $\ell \leq \frac{\log_2 n}{(5+\epsilon)\cdot\lceil \log_2 \sigma \rceil}$, find the position $j_{\max} = \max(\{j' \in [j, j + \ell) \mid T[j'..n] \prec T[i..n]\} \cup \{j - 1\})$.*

Now we have all the tools needed to describe the algorithm. We start with an empty stack $\mathcal{Q}$ and $\mathcal{B} = (,$ i.e., with the opening parenthesis of the artificial root node 0. Now we process the blocks $B_1, \ldots, B_N$ in left-to-right order. At the time at which we start processing

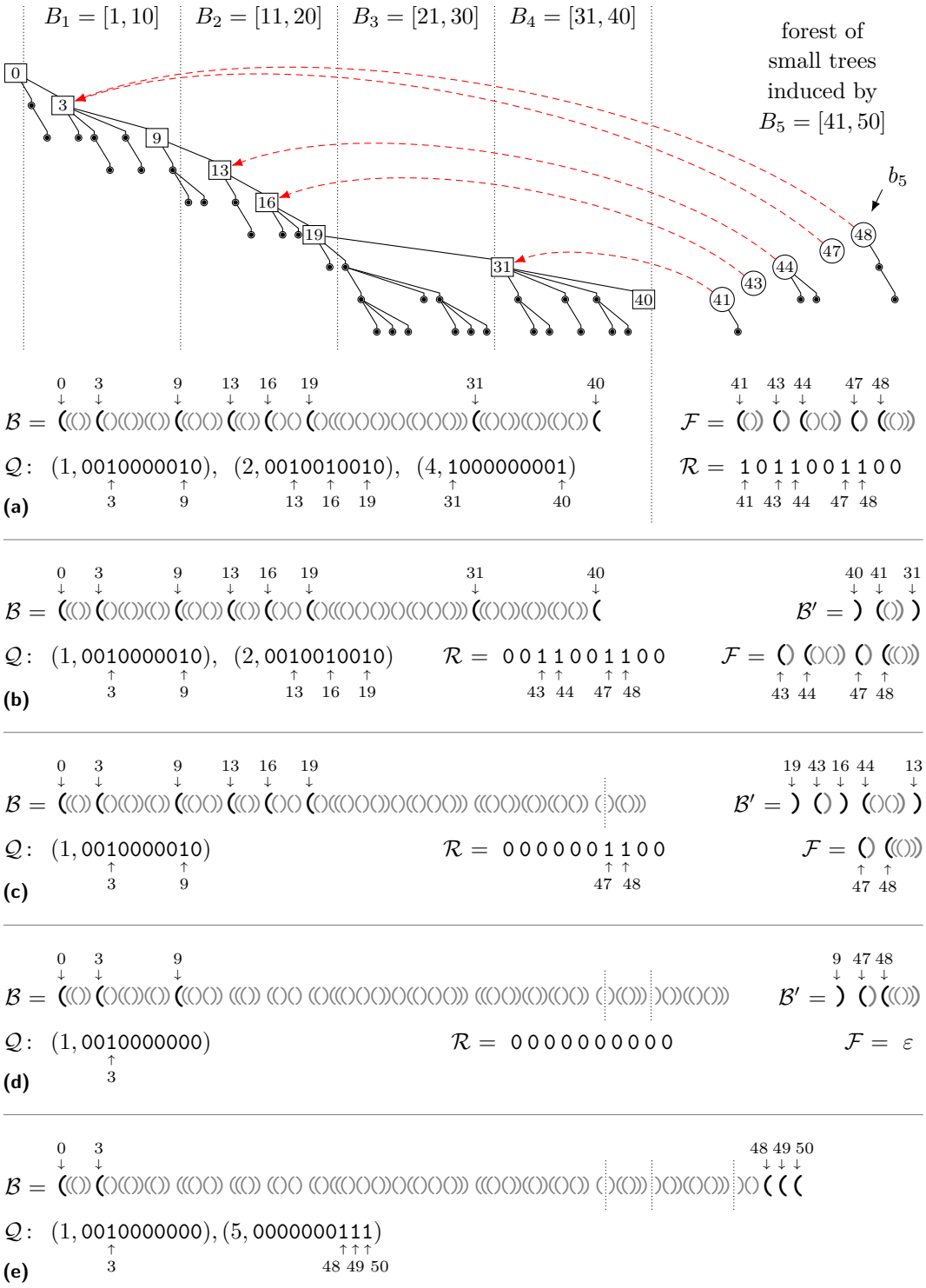



**Figure 2** Data structures during the execution of the blockwise algorithm with block size $k = 10$. While processing $B_5 = [41..50]$, the dashed edges will be inserted into the partial PSS tree induced by $[0..40]$. The drawings show the state of the relevant data structures before calling the subroutine during the first iteration of the main routine (a), and after calling the subroutine in the first (b), second (c), and third (d) iteration of the main routine. The state after finalizing $B_5$ is shown in (e).

$B_x$, the stack $\mathcal{Q}$ contains the nodes on the path from $xk-k$ to 0 in the blockwise manner described above, and $\mathcal{B}$ contains the prefix of the BPS of the PSS tree that ends with the opening parenthesis of node $xk-k$. We start by querying Lemma 3 with $B_x$ and obtain the BPS $\mathcal{F}$ of the forest induced by $B_x$, as well as the bitvector $\mathcal{R}$ indicating the small roots. We then find the rightmost 1-bit in $\mathcal{R}$ in constant time (there are only $2^k = \mathcal{O}(\sqrt[8]{n})$ possible values of $\mathcal{R}$, hence a lookup table for rightmost or leftmost 1-bits can be precomputed in $o(n/\log n)$ time). If this bit is at position $\mathcal{R}[b'_x]$, then $b_x = b'_x + xk - k$ is the rightmost small root in the forest induced by $B_x$. Note that $T[b_x..n]$ is the lexicographically smallest suffix starting in $B_x$. The state after this step is visualized in Figure 2a. Now we repeatedly run the interleaving main routine described below, during which we will alter $\mathcal{F}$, $\mathcal{R}$, $\mathcal{Q}$, and $\mathcal{B}$.

**Main Routine.** The goal of this routine is to interleave (the remaining small trees of) $B_x$ with the topmost block on the stack. If $\mathcal{F}$ is empty (which happens if and only if $\mathcal{R}$ contains only zeroes), then we have attached all small trees and the main routine terminates. Otherwise, if $\mathcal{Q}$ is empty, the remaining small trees need to be attached to the root of the PSS tree, and we append $\mathcal{F}$ to $\mathcal{B}$. This takes $\mathcal{O}(1)$ time and also terminates the main routine.

If neither $\mathcal{F}$ nor $\mathcal{Q}$ are empty, then we retrieve and pop the topmost pair $(y, \mathcal{L})$ from $\mathcal{Q}$. We use Lemma 4 to obtain $b_y = \max(\{j' \in B_y \mid T[j'..n] \prec T[b_x..n]\} \cup \{yk-k\})$, and the corresponding within-block offset $b'_y = b_y - yk + k$. If $\mathsf{pss}[b_x] \in B_y$ then $b_y = \mathsf{pss}[b_x]$, and all remaining small trees have to be attached to nodes from $B_y \cap [b_y, n]$. Since $b_x$ will be attached to $b_y$, none of the nodes from $B_y \cap (b_y, n]$ will remain on the stack. Hence we compute a bitvector $\mathcal{L}'[1..k]$ where for $j' \in [1,k]$ it holds $\mathcal{L}'[j'] = 1$ if and only if $\mathcal{L}[j'] = 1$ and $j' \leq b'_y$ (this takes constant time using bitwise operations). We then push $(y, \mathcal{L}')$ back onto the stack. If however $\mathsf{pss}[b_x] \notin B_y$, then $b_y = yk - k$ (the first position to the left of $B_y$) and $b_x$ will be attached to a node in a block left of $B_y$. This means that block $B_y$ will no longer be on the stack. Note that either way $\forall j' \in (b_y, b_x) : T[j'..n] \succ T[b_x..n]$.

Our next task is as follows. We have to attach some (possibly none, possibly all) of the remaining small trees to nodes in $B_y$. We reflect this change in $\mathcal{F}$ and $\mathcal{R}$ by removing the corresponding prefix of $\mathcal{F}$, and setting the corresponding bits in $\mathcal{R}$ to 0. Simultaneously, we extend $\mathcal{B}$ such that it contains the newly attached small trees, possibly interleaved with additional closing parentheses of nodes from $B_y$. This is realized by the following interleaving subroutine, which we run in a loop (and which will later be replaced by a single constant time table lookup). A sequence $\mathcal{B}'$ is used to buffer the parentheses that we will append to $\mathcal{B}$.

**Subroutine.** If either $\mathcal{L}$ or $\mathcal{R}$ consists only of 0-bits, we terminate the subroutine. Otherwise, we obtain the rightmost 1-bit of $\mathcal{L}$ (with a lookup table). If this bit is at position $\mathcal{L}[j']$, then the corresponding absolute position is $j = j' + yk - k$. If $j' = b'_y$ (which is equivalent to $j = b_y = \mathsf{pss}[b_x]$), then all remaining small trees need to be attached to $j$, and we append $\mathcal{F}$ to $\mathcal{B}'$. We replace $\mathcal{F}$ with $\varepsilon$ and $\mathcal{R}$ with an all-zero bitvector. This terminates the subroutine.

Otherwise (i.e., if $j' > b'_y$ or equivalently $j > b_y$), we obtain the leftmost 1-bit of $\mathcal{R}$ (with a lookup table). If this bit is at position $\mathcal{R}[i']$, then $i = i' + xk - k$ is the leftmost small root that we still have to attach. Now we have to determine if $T[i..n] \prec T[j..n]$. (This state is equivalent to reaching the head of the inner loop of Algorithm 1 with the current values of $i$ and $j$.) It holds $T[i..n] \prec T[j..n]$ if and only if $T[i..b_x) \preceq T[j..j + b_x - i)$. This is because $j + b_x - i \in (b_y, b_x)$, and hence we have already established that $T[b_x..n] \prec T[j + b_x - i..n]$. Thus, if $T[i..b_x) = T[j..j + b_x - i)$, it immediately follows that $T[i..n] \prec T[j..n]$. Note that $T[i..b_x)$ and $T[j..j + b_x - i)$ are substrings of $T(xk-k..xk]$ and $T(yk-k..yk+k]$ respectively, which will later be relevant for an efficient implementation. If $T[i..n] \prec T[j..n]$, then we

append ) to $\mathcal{B}'$ (this is the closing parenthesis of node $j$), and we assign $\mathcal{L}[j'] = 0$. If, however, $T[i..n] \succ T[j..n]$, then $\mathsf{pss}[i] = j$. In this case, we take the prefix of $\mathcal{F}$ that corresponds to the small tree rooted in $i$ (which is the shortest balanced prefix of $\mathcal{F}$), and append it to $\mathcal{B}'$. We remove this prefix from $\mathcal{F}$ and assign $\mathcal{R}[i'] = 0$. We then continue with the next iteration of the subroutine. After the subroutine terminates, we append $\mathcal{B}'$ to $\mathcal{B}$ and continue with the next iteration of the main routine. Figures 2b–2d shows the result of the subroutine in three consecutive iterations of the main routine.

**Finalizing the Block.** Once the main routine terminates for block $B_x$, we have attached all the small trees of $B_x$ to $\mathcal{B}$. The stack $\mathcal{Q}$ contains the blockwise representation of all the nodes on the path from $xk$ to 0, except for the ones in block $B_x$. Before we can continue with the next iteration of the main routine, we have to push $(x, \mathcal{L}'')$ on the stack, where the 1-bits in $\mathcal{L}''$ correspond to the nodes on the path from $xk$ to $b_x$. Note that this information can be obtained from the state of $\mathcal{F}$ at the beginning of the main routine iteration. Since $\mathcal{F}$ is a bitvector of length $2k$, a lookup table $W[1..2^{2k}]$ suffices to store the bitvector $\mathcal{L}''$ for each possible $\mathcal{F}$. The table has $\mathcal{O}(\sqrt[4]{n})$ entries and can be filled naively in $\mathcal{O}(\sqrt[4]{n} \cdot \mathrm{polylog}(n)) \subset \mathcal{O}(n/\log n)$ time. Once we need $\mathcal{L}''$, we simply lookup $W[\mathsf{int}(\mathcal{F})]$ in constant time. Finally, in order to continue, the last written parenthesis needs to be the opening parenthesis of $xk$. Hence we remove the at most $k$ trailing closing parentheses of $\mathcal{B}$ (in constant time, using another lookup table), and then continue by processing block $B_{x+1}$. Figure 2e shows the running example after finalizing the processed block.

After block $B_N$ has been processed, we finish the algorithm execution by appending the $2n + 2 - |\mathcal{B}|$ closing parentheses of the nodes on the path from $n$ to 0. This can be done in $\mathcal{O}(n/\log n)$ time by appending them one word (rather than one parenthesis) at a time.

## 3.2 Analyzing the Time Complexity

The initial and final processing of each block (i.e., computing $\mathcal{F}$, $\mathcal{R}$, $b_x$, and the pair $(x, \mathcal{L}'')$ to push on the stack) takes constant time. There are exactly $N$ terminal iterations of the main routine, i.e., iterations where either $\mathcal{F}$ or $\mathcal{Q}$ is empty. Each terminal iteration takes constant time. In each of the non-terminal iterations, we pop a pair $(y, \mathcal{L})$ from the stack. If we do not push an updated pair $(y, \mathcal{L}')$ back onto the stack, then block $B_y$ will never participate in the stack again, and hence this case occurs at most $N$ times. If, however, we do push an updated pair $(y, \mathcal{L}')$ back onto the stack, then during the same main routine iteration we will also attach all remaining small trees of $B_x$ to the partial PSS tree, which can also occur only $N$ times. Hence the total number of iterations of the main routine is $\mathcal{O}(N)$. In each non-terminal iteration of the main routine, we call the subroutine exactly once (even though a single call may lead to multiple iterations of the subroutine). Apart from this call, each iteration of the main routine takes constant time.

It remains to be shown how to implement the subroutine such that the $\mathcal{O}(N)$ calls take $\mathcal{O}(n/\log_\sigma n)$ time in total. A straightforward naive implementation takes $\mathcal{O}(\mathrm{poly}(k)) \subseteq \mathcal{O}(\mathrm{polylog}(n))$ time per call. Note that the subroutine only accesses the following information: $\mathcal{L}$, $b'_y$, $\mathcal{R}$ (which allows access to $b'_x$), $\mathcal{F}$, and substrings $T_y = T(yk - k..yk + k]$ and $T_x = T(xk - k..xk]$. Bitvectors $\mathcal{L}$ and $\mathcal{R}$ are of length $k$ bits each; $b'_y$ is an integer from $[0, k]$ and hence can be encoded in $\lceil \log_2(k+1) \rceil \leq \lfloor 0.99k \rfloor$ bits (for sufficiently large $k$); sequence $\mathcal{F}$ is of length at most $2k$ bits; strings $T_y$ and $T_x$ in packed representation require $2k \lceil \log_2 \sigma \rceil$ and $k \lceil \log_2 \sigma \rceil$ bits respectively. This motivates a lookup table

$$M[1..2^k][1..2^{\lfloor 0.99k \rfloor}][1..2^k][1..2^{2k}][1..2^{2k\lceil \log_2 \sigma \rceil}][1..2^{k\lceil \log_2 \sigma \rceil}].$$

In entry $M[\mathsf{int}(\mathcal{L})][b'_y][\mathsf{int}(\mathcal{R})][\mathsf{int}(\mathcal{F})][\mathsf{int}(T_y)][\mathsf{int}(T_x)]$, we store $\mathcal{B}'$ as well as the new values of $\mathcal{R}$ and $\mathcal{F}$ after running the subroutine. Note that $\mathcal{B}'$ is of length at most $3k$ because it contains at most all the parentheses from $\mathcal{F}$ and one closing parenthesis per 1-bit in $\mathcal{L}$. Hence the information stored in each table entry fits in a constant number of words, and can be retrieved in constant time. We fill the table in a lazy manner. Initially, we mark each entry as uninitialized. When accessing $M[\mathsf{int}(\mathcal{L})][b'_y][\mathsf{int}(\mathcal{R})][\mathsf{int}(\mathcal{F})][\mathsf{int}(T_y)][\mathsf{int}(T_x)]$, we check if this entry is marked. If it is, then we run the naive $\mathcal{O}(\mathrm{polylog}(n))$ time algorithm for the subroutine, store the result in the entry, and remove its marking. Otherwise, the entry already contains the values of $\mathcal{B}'$, $\mathcal{R}$, and $\mathcal{F}$ after running the subroutine, and we return them in constant time. The lookup table has at most $2^{7.99k\lceil\log_2\sigma\rceil} \leq 2^{\log_2 n \cdot 7.99/8} = n^{7.99/8}$ entries. Computing one entry takes $\mathcal{O}(\mathrm{polylog}(n))$ time. Thus, the entire time spent on filling the table (i.e., on running the subroutine naively) is $\mathcal{O}(n^{7.99/8} \cdot \mathrm{polylog}(n)) \subset \mathcal{O}(n/\log n)$. Additional $\mathcal{O}(n/\log_\sigma n)$ preprocessing time is needed for Lemmas 1, 3, and 4.

## 4 Proving Lemma 3

A key insight for the proof of Lemma 3 is that the lexicographical order of suffixes starting in a small range almost entirely depends on a short substring. This is formally expressed by the auxiliary lemma below.

▶ **Lemma 5.** *Let $T[1..n]$ be a string over a totally ordered alphabet, and let $[i, i+2\ell) \subseteq [1, n]$ be a non-empty interval of even length. Then at least one of the following properties holds:*

- $\forall x, y \in [i, i+\ell) : T[x..n] \prec T[y..n] \iff T[x..i+2\ell) \prec T[y..i+2\ell)$, *or*
- $\forall x, y \in [i, i+\ell) : T[x..n] \prec T[y..n] \iff T[x..i+2\ell)\# \prec T[y..i+2\ell)\#$,

  *where $\#$ is an infinitely large symbol, i.e., $\forall i' \in [1, n] : T[i'] < \#$.*

**Proof.** Let $\tilde{n} = i + 2\ell$. Assume that the lemma does not hold, then there are indices $x_1, x_2, y_1, y_2 \in [i, i+\ell)$ such that $T[x_1..n] \prec T[y_1..n]$ but $T[x_1..\tilde{n}) \succ T[y_1..\tilde{n})$, and $T[x_2..n] \prec T[y_2..n]$ but $T[x_2..\tilde{n})\# \succ T[y_2..\tilde{n})\#$. It is easy to see that this implies

$$\begin{aligned} T[y_1..\tilde{n}) &\prec T[x_1..\tilde{n}) \prec T[x_1..n] \prec T[y_1..n] = T[y_1..\tilde{n})T[\tilde{n}..n], \text{ and} \\ T[x_2..\tilde{n})\# &\succ T[y_2..\tilde{n})\# \succ T[y_2..n] \succ T[x_2..n] = T[x_2..\tilde{n})T[\tilde{n}..n]. \end{aligned}$$

Due to first condition, $T[y_1..\tilde{n})$ is a proper prefix of $T[x_1..\tilde{n})$ and it holds $x_1 < y_1$. Note that $T[y_1..\tilde{n})$ is therefore also a proper suffix (and hence a border) of $T[x_1..\tilde{n})$, and thus $T[x_1..\tilde{n})$ has period $p_1 = (y_1 - x_1)$. By the same reasoning, the second condition implies that $T[x_2..\tilde{n})$ is a border of $T[y_2..\tilde{n})$. Hence $y_2 < x_2$, and $T[y_2..\tilde{n})$ has period $p_2 = (x_2 - y_2)$. By combining these observations with the initial assumption, we obtain

$$\begin{aligned} T[y_1..\tilde{n})T[\tilde{n}-p_1..n] &= T[x_1..n] \prec T[y_1..n] = T[y_1..\tilde{n})T[\tilde{n}..n], \quad \text{and} \\ T[x_2..\tilde{n})T[\tilde{n}..n] &= T[x_2..n] \prec T[y_2..n] = T[x_2..\tilde{n})T[\tilde{n}-p_2..n]. \end{aligned}$$

The former implies $T[\tilde{n}-p_1..n] \prec T[\tilde{n}..n]$, the latter implies $T[\tilde{n}..n] \prec T[\tilde{n}-p_2..n]$. Hence

$$T[\tilde{n}-p_1..\tilde{n})T[\tilde{n}..n] \prec T[\tilde{n}..n] \prec T[\tilde{n}-p_2..\tilde{n})T[\tilde{n}..n].$$

Since $T[\max(x_1, y_2)..\tilde{n})$ is a suffix of both $T[x_1..\tilde{n})$ and $T[y_2..\tilde{n})$, it has periods $p_1$ and $p_2$. Note that $x_1 < i + \ell - p_1$ and $y_2 < i + \ell - p_2$, and hence $T[\max(x_1, y_2)..\tilde{n})$ is of length $\tilde{n} - \max(x_1, y_2) > \tilde{n} - i - \ell + \min(p_1, p_2) = \ell + \min(p_1, p_2) > p_1 + p_2$. Therefore, it follows from the periodicity lemma [18] that $T[\max(x_1, y_2)..\tilde{n})$ has period $p_0 = \gcd(p_1, p_2)$. Since

both $T[\tilde{n}-p_1..\tilde{n})$ and $T[\tilde{n}-p_2..\tilde{n})$ are suffixes of $T[\max(x_1,y_2)..\tilde{n})$, they also have period $p_0$. Let $\alpha = T[\tilde{n}-p_0..\tilde{n})$, $k_1 = p_1/p_0$ and $k_2 = p_2/p_0$. Then both $k_1$ and $k_2$ are positive integers, and it holds $T[\tilde{n}-p_1..\tilde{n}) = \alpha^{k_1}$ and $T[\tilde{n}-p_2..\tilde{n}) = \alpha^{k_2}$. Let $k_0$ be the largest integer (possibly 0) such that $T[\tilde{n}..n] = \alpha^{k_0}T[\tilde{n}+k_0p_0..n]$, and let $\beta = T[\tilde{n}+k_0p_0..n]$. Then $\alpha$ is not a prefix of $\beta$. The inequality above can be written as $\alpha^{k_1+k_0}\beta \prec \alpha^{k_0}\beta \prec \alpha^{k_2+k_0}\beta$. However, this is equivalent to $\alpha^{k_1}\beta \prec \beta \prec \alpha^{k_2}\beta$, which implies that $\alpha$ is a prefix of $\beta$. Due to this contradiction, the initial assumption must be false, and the lemma holds. ◀

Now we are ready to show Lemma 3, which is restated below.

▶ **Lemma 3.** *Let $T \in [0,\sigma)^n$ be a string in packed representation and let $\epsilon \in \mathbb{R}^+$. After $\mathcal{O}(n/\log_\sigma n)$ preprocessing time, the following type of query can be answered in $\mathcal{O}(1)$ time. Given a range $[i,i+\ell) \subseteq [1,n]$ of length $\ell \leq \frac{\log_2 n}{(2+\epsilon)\lceil\log_2\sigma\rceil}$, output the BPS of the PSS forest induced by $[i,i+\ell)$, as well as a bitvector $\mathcal{R}[1..\ell]$ such that for $j \in [1,\ell]$ it holds $\mathcal{R}[j] = 1$ if and only if $i+j-1$ is the root of a tree in the forest.*

**Proof.** The answer to any query $[i,i+\ell)$ is a parentheses sequence of length exactly $2\ell$ and a bitvector of length $\ell$. Hence it fits in a constant number of words. Let $\ell_{\max} = \left\lfloor \frac{\log_2 n}{(2+\epsilon)\cdot\lceil\log_2\sigma\rceil} \right\rfloor$. We precompute a 2D lookup table $E[1..2\ell_{\max}][1..\ell_{\max}]$ with the purpose of answering the subset of queries that satisfy $i+2\ell_{\max} > n$. For any such query $[i,i+\ell)$, it holds $n-i+1 \in [1,2\ell_{\max}]$, and we explicitly store its answer in $E[n-i+1][\ell]$. Since these queries only consider suffixes of length $\mathcal{O}(\log n)$, each of the $\mathcal{O}(\log^2 n)$ table entries can be computed naively in $\mathcal{O}(\text{polylog}(n))$ time.

We answer the remaining queries using the LCE data structure from Lemma 1 and additional lookup tables. For each possible value of $\ell$, we construct tables $A_\ell[1..2^{2\ell\lceil\log_2\sigma\rceil}]$ and $B_\ell[1..2^{2\ell\lceil\log_2\sigma\rceil}]$. For every string $S \in [0,\sigma)^{2\ell}$, we store at position $A_\ell[\mathsf{int}(S)]$ the BPS of the PSS forest of $S$ that is induced by $[1,\ell]$, as well as the bitvector that indicates the roots. At position $B_\ell[\mathsf{int}(S)]$, we store the BPS of the PSS forest of $S\#$ that is induced by $[1,\ell]$, as well as the bitvector that indicates the roots.

When answering query $[i,i+\ell)$, we first extract $T' = T[i,i+2\ell)$. Due to Lemma 5, the answer to the query is either $A_\ell[\mathsf{int}(T')]$ or $B_\ell[\mathsf{int}(T')]$. A table $C_\ell[1..2^{2\ell\lceil\log_2\sigma\rceil}]$ is used to decide which answer is correct. For every string $S \in [0,\sigma)^{2\ell}$, we store at position $C_\ell[\mathsf{int}(S)]$ an integer pair $(x,y) \in [1,\ell]^2$ such that $S[x..2\ell] \prec S[y..2\ell]$ and $S[x..2\ell]\# \succ S[y..2\ell]\#$ (or $x=y=1$ if such a pair does not exist). This is as a witness pair of suffixes for which $S$ and $S\#$ disagree on the lexicographical order. At query time, we lookup $(\hat{x},\hat{y}) = C_\ell[T']$. If $T[i+\hat{x}-1..n] \prec T[i+\hat{y}-1..n]$, then we return $A_\ell[\mathsf{int}(T')]$, and otherwise we return $B_\ell[\mathsf{int}(T')]$. The correctness follows from Lemma 5. Testing $T[i+\hat{x}-1..n] \prec T[i+\hat{y}-1..n]$ takes constant time with the LCE data structure from Lemma 1. Extracting $T'$ and performing table lookups also takes constant time because $T'$ fits in a single word of memory.

A single lookup table entry can be computed naively in $\mathcal{O}(\text{polylog}(n))$ time. There are $\mathcal{O}(\log n)$ tables, each storing at most $2^{2\ell_{\max}\lceil\log_2\sigma\rceil} \leq 2^{\log_2 n/(1+\epsilon/2)} = \sqrt[1+\epsilon/2]{n}$ entries. Thus, the precomputation of lookup tables takes $\mathcal{O}(\sqrt[1+\epsilon/2]{n}\cdot\text{polylog}(n))$ time, which is dominated by the $\mathcal{O}(n/\log_\sigma n)$ time needed to construct the LCE data structure. ◀

## 5 Proving Lemma 4

The proof of Lemma 4 relies on the properties of periodic substrings that are stated below.

▶ **Proposition 6.** *Let $\alpha$, $\beta$, and $\gamma$ be arbitrary strings. The following properties hold.*

1. *If $\alpha\beta \succ \beta$ and $\alpha\gamma \prec \gamma$ then $\beta \prec \gamma$.*
2. *If $\alpha\gamma \prec \gamma$ and $\alpha$ is not a prefix of $\gamma$, then $\forall x,y \in \mathbb{N}^0 : x > y \implies \alpha^x\beta \prec \alpha^y\gamma$.*

**Proof.** We start with (1). Let $k \in \mathbb{N}^0$ be the maximal value such that both $\beta = \alpha^k\beta'$ and $\gamma = \alpha^k\gamma'$ for some (possibly empty) strings $\beta'$ and $\gamma'$. If $\alpha^{k+1}\beta' \succ \alpha^k\beta'$ and $\alpha^{k+1}\gamma' \prec \alpha^k\gamma'$, then $\beta' \prec \alpha\beta'$ and $\alpha\gamma' \prec \gamma'$. Now assume that $\gamma' \preceq \beta'$, then $\alpha\gamma' \prec \gamma' \preceq \beta' \prec \alpha\beta'$. However, this implies that $\alpha$ is a prefix of both $\beta$ and $\gamma$, which contradicts the definition of $k$. Thus $\beta' \prec \gamma'$, which also implies $\beta = \alpha^k\beta' \prec \alpha^k\gamma' = \gamma$. For (2), consider any $x, y \in \mathbb{N}^0$ with $x > y$, and assume that $\alpha\gamma \prec \gamma$. Since $\alpha$ is not a prefix of $\gamma$, it follows from $\alpha\gamma \prec \gamma$ that $\alpha\delta \prec \gamma$ for every string $\delta$. Hence also $\alpha^{x-y}\beta \prec \gamma$, which implies $\alpha^x\beta = \alpha^y\alpha^{x-y}\beta \prec \alpha^y\gamma$. ◀

Now we are ready to show Lemma 4, which is restated below.

▶ **Lemma 4.** *Let $T \in [0,\sigma)^n$ be a string in packed representation and let $\epsilon \in \mathbb{R}^+$. After $\mathcal{O}(n/\log_\sigma n)$ preprocessing time, we can answer the following type of query in $\mathcal{O}(1)$ time. Given a position $i \in [1,n]$ and a non-empty interval $[j, j+\ell) \subseteq [1,n]$ of length $\ell \leq \frac{\log_2 n}{(5+\epsilon)\cdot\lceil\log_2\sigma\rceil}$, find the position $j_{\max} = \max(\{j' \in [j, j+\ell) \mid T[j'..n] \prec T[i..n]\} \cup \{j-1\})$.*

**Proof.** Similarly to what was done in Lemma 3, we spend $\mathcal{O}(\text{polylog}(n))$ time to precompute the answers to all queries that satisfy $j + 3\ell \geq n$ or $i + 2\ell \geq n$. For any of the remaining queries, we consider the set

$$C = \{j' \in [j, j+\ell) \mid T[j'..j'+2\ell) = T[i..i+2\ell)\} = \{c_1, c_2, \ldots, c_h\}$$

with $c_1 < c_2 < \cdots < c_h$. This set contains exactly the positions $j' \in [j, j+\ell)$ for which we cannot easily determine whether $T[j'..n] \prec T[i..n]$ by inspecting only a small number of symbols. Hence it captures the difficult part of answering a query, and we treat it separately from the rest. We answer the query using the following subsets of $[j, j+\ell)$:

- $D' = \{j' \in C \mid T[j'..n] \prec T[i..n]\}$ (the hard subset), and
- $D'' = \{j' \in [j, j+\ell) \setminus C \mid T[j'..n] \prec T[i..n]\}$ (the easy subset).

The result of the query is $j_{\max} = \max(j'_{\max}, j''_{\max})$, where $j'_{\max} = \max(D' \cup \{j-1\})$ and $j''_{\max} = \max(D'' \cup \{j-1\})$. We start with the significantly harder task of computing $j'_{\max}$. First, we outline the algorithmic approach and the combinatorial properties of the present substrings (without giving details of an efficient implementation). Later, we describe lookup tables that achieve the claimed preprocessing and query times.

**Periodicity of $T[i..i+2\ell)$ and $T[c_1..c_h+2\ell)$.** We show that, if $|C| \geq 2$, then there is some $p$ such that $T[c_1..c_h+2\ell)$ has period $p$, and $\forall x \in [1,h) : c_{x+1} - c_x = p$. This is similar to [34, Lemma 1] and [28, Lemma 2]. Assume that $|C| \geq 2$. For $x \in [1,h)$, let $p_x = c_{x+1} - c_x < \ell$. By design of $C$, it holds $T[c_x..c_x+2\ell-p_x) = T[c_{x+1}..c_{x+1}+2\ell-p_x) = T[c_x+p_x..c_x+2\ell)$. This means that $T[i..i+2\ell) = T[c_x..c_x+2\ell)$ has a border of length $2\ell - p_x$, and therefore it has period $p_x$. Let $p$ be the smallest period of $T[i..i+2\ell)$. If there was some $x \in [1,h)$ such that $p_x < p$, then $p$ would not be the smallest period of $T[i..i+2\ell)$. Hence $p_x \geq p$. Now we show that $\forall x \in [1,h) : p_x = p$. For the sake of contradiction, assume $p_x > p$. By definition of $C$, it holds $p_x < \ell$, which means that $T[c_x..c_x+2\ell)$ and $T[c_{x+1}..c_{x+1}+2\ell)$ overlap by $c_x + 2\ell - c_{x+1} = 2\ell - p_x > \ell > p$ symbols. Due to this overlap, and because the identical substrings $T[c_x..c_x+2\ell)$ and $T[c_{x+1}..c_{x+1}+2\ell)$ have period $p$, it is clear that also their union $T[c_x..c_{x+1}+2\ell)$ has period $p$. However, this implies $T[c_x..c_x+2\ell) = T[c_x+p..c_x+p+2\ell)$, which means that $c_x + p$ should be in $C$. Due to this contradiction, it holds $p_x = p$. It also follows that $T[c_1..c_h+2\ell)$ has period $p$.

**Computing $j'_{\max}$ from $c_1$, $c_h$, and $p$.** We will later introduce lookup tables that output $c_1$, $c_h$, and $p$ for any query in constant time. The tables might return that $c_1$ and $c_h$ do not exist (i.e., $|C| = 0$), in which case we report $j'_{\max} = j - 1$. Otherwise, it might be that $c_1 = c_h$ (i.e., $|C| = 1$). In this case, we report that $j'_{\max} = c_1$ if $T[i..n] \succ T[c_1..n]$ (using an LCE query for the comparison). Otherwise, we report $j'_{\max} = j - 1$. It remains to be shown how to compute $j'_{\max}$ if $c_1 \neq c_h$ (i.e., if $|C| \geq 2$, and the previously described periodicity exists).

We evaluate $T[i..n] \prec T[i + p..n]$ and $T[c_h..n] \prec [c_h + p..n]$ (using LCE queries). Due to the periodicity of $T[c_1..c_h + 2\ell)$, for $x \in [1, h)$ it holds $T[c_h..n] \prec T[c_h + p..n]$ if and only if

$$T[c_x..n] \;=\; T[i..i+p)^{k-x}T[c_h..n] \;\prec\; T[i..i+p)^{k-x}T[c_h+p..n] \;=\; T[c_{x+1}..n].$$

Hence either $T[c_1..n] \prec T[c_2..n] \prec \cdots \prec T[c_h..n]$ or $T[c_1..n] \succ T[c_2..n] \succ \cdots \succ T[c_h..n]$, and we already know which of the two applies. Depending on the outcome of the lexicographical comparisons, we report $j'_{\max}$ according to one of the following three cases.

**Case 1:** $T[c_1..n] \succ T[c_2..n] \succ \cdots \succ T[c_h..n]$.

For the computation of $j'_{\max}$, we are only interested in the rightmost $x \in [1, h]$ such that $T[i..n] \succ T[c_x..n]$. Since $T[c_h..n]$ is both rightmost and lexicographically minimal among all the possible $T[c_x..n]$, we simply use another LCE query to check if $T[i..n] \succ T[c_h..n]$. If yes, we report $j'_{\max} = c_h$. Otherwise, we report $j'_{\max} = j - 1$.

**Case 2:** $T[i..n] \succ T[i + p..n]$ and $T[c_1..n] \prec T[c_2..n] \prec \cdots \prec T[c_h..n]$.

Let $\alpha = T[i..i + p)$, $\beta = T[i + p..n]$, and $\gamma = T[c_2..n]$. The precondition of this case means that $\alpha\beta \succ \beta$ and $\alpha\gamma \prec \gamma$. Proposition 6.1 implies $\beta \prec \gamma$, and thus also $T[i..n] = \alpha\beta \prec \alpha\gamma = T[c_1..n] \prec T[c_2..n] \prec \cdots \prec T[c_h..n]$. Hence we report $j'_{\max} = j - 1$.

**Case 3:** $T[i..n] \prec T[i + p..n]$ and $T[c_1..n] \prec T[c_2..n] \prec \cdots \prec T[c_h..n]$.

Let $\alpha = T[i..i+p)$. We compute $r = \lfloor \textsc{lce}(i, i + p)/p \rfloor + 1$ and $s = \lfloor \textsc{lce}(c_1, c_1 + p)/p \rfloor + 1$, i.e., the respectively maximal integer powers with $T[i..n] = \alpha^r\beta$ and $T[c_1..n] = \alpha^s\gamma$, where $\beta = T[i + rp..n]$ and $\gamma = T[c_1 + sp..n]$. The precondition of this case means that $\alpha^s\gamma \prec \alpha^{s-1}\gamma$, and thus also $\alpha\gamma \prec \alpha$. Note that $\alpha$ is not a prefix of $\gamma$. Hence Proposition 6.2 implies that $\alpha^r\beta \prec \alpha^{s'}\gamma$ for any $s' < r$. Thus, for $x \in [1, h]$, if $s-x+1 < r$ then $T[i..n] = \alpha^r\beta \prec \alpha^{s-x+1}\gamma = T[c_x..n]$. Hence we only have to consider $x \leq s - r + 1$. On the other hand, the precondition of the case also implies $\alpha^r\beta \prec \alpha^{r-1}\beta$, and thus $\alpha\beta \prec \alpha$. Also, $\alpha$ is not a prefix of $\beta$. Hence Proposition 6.2 (with swapped roles of $\beta$ and $\gamma$) implies that $\alpha^r\beta \succ \alpha^{s'}\gamma$ for any $s' > r$. For $x \in [1, h]$, if $x \leq s - r$ then $T[i..n] = \alpha^r\beta \succ \alpha^{s-x+1}\gamma = T[c_x..n]$.
This motivates the following strategy. If $s - r + 1 < 1$, then there is no suitable choice of $x$ and we report $j'_{\max} = j - 1$. If $k \leq s - r$, then $T[i..n] \succ T[c_h..n]$ and we report $j'_{\max} = c_h$. We are left with the case where $s - r + 1 \in [1, h]$. If $T[i..n] \succ T[c_{s-r+1}..n]$, then we report $j'_{\max} = c_{s-r+1}$ (we use another LCE query to achieve constant time). If we still have not reported anything, then we report $j'_{\max} = c_{s-r}$ if and only if $s - r \in [1, h]$ (we have already established that $T[i..n] \succ T[c_{s-r}..n]$). If, however, $s - r \notin [1, h]$, then we report $j'_{\max} = j - 1$.

The three cases are exhaustive, and it takes constant time to determine which case applies. Regardless of the case, we report $j'_{\max}$ in constant time. We require the LCE data structure from Lemma 1, and hence the preprocessing time is $\mathcal{O}(n/\log_\sigma n)$.

**Lookup Tables for $c_1$, $c_h$, $p$, and $j''_{\max}$.** As described above, we can compute $j'_{\max}$ in constant time if we can determine $c_1$, $c_h$, and $p$ in constant time. Note that these values depend solely on the substrings $T[i..i+2\ell)$ and $T[j..j+3\ell)$. This also holds for $j''_{\max}$, which can be written as $j''_{\max} = \max(\{j' \in [j, j+\ell) \mid T[j'..j'+2\ell) \prec T[i..i+2\ell)\} \cup \{j-1\})$. For each possible value of $\ell$, we compute a lookup table $L_\ell[1..2^{2\ell\lceil\log_2\sigma\rceil}][1..2^{3\ell\lceil\log_2\sigma\rceil}]$. Let $S_1 \in [0,\sigma)^{2\ell}$ and $S_2 \in [0,\sigma)^{3\ell}$ be packed strings. In entry $L_\ell[\mathsf{int}(S_1)][\mathsf{int}(S_2)]$, we store the quadruple $\langle \hat{p}, \hat{c}_{\min}, \hat{c}_{\max}, \hat{y}_{\max}\rangle$, where

- $\hat{p}$ is the shortest period of $S_1$,
- $\hat{c}_{\min} = \min(\{x' \in [1,\ell] \mid S_2[x'..x'+2\ell) = S_1\} \cup \{\infty\})$,
- $\hat{c}_{\max} = \max(\{x' \in [1,\ell] \mid S_2[x'..x'+2\ell) = S_1\} \cup \{-\infty\})$,
- $\hat{y}_{\max} = \max(\{x' \in [1,\ell] \mid S_2[x'..x'+2\ell) \prec S_1\} \cup \{-\infty\})$.

A single entry $L_\ell[\mathsf{int}(S_1)][\mathsf{int}(S_2)]$ can be computed naively in $\mathcal{O}(\mathrm{poly}(\ell)) \subseteq \mathcal{O}(\mathrm{polylog}(n))$ time. Table $L_\ell$ has $2^{5\ell\lceil\log_2\sigma\rceil} \leq 2^{\log_2 n/(1+\epsilon/5)} = \sqrt[1+\epsilon/5]{n}$ entries, and there are $\mathcal{O}(\log n)$ tables. Thus, the entire preprocessing time is $\mathcal{O}(\sqrt[1+\epsilon/5]{n} \cdot \mathrm{polylog}(n)) \subset \mathcal{O}(n/\log_\sigma n)$. Whenever we have to answer a query $i$, $[j, j+\ell)$, we extract $T' = T[i..i+2\ell)$ and $T'' = T[j..j+3\ell)$ and lookup $\langle p, c_{\min}, c_{\max}, y_{\max}\rangle = L_\ell[\mathsf{int}(T')][\mathsf{int}(T'')]$. This takes constant time because $T'$ and $T''$ fit in a single word of memory. From the construction of $L_\ell$, it is clear that

- $p$ is the shortest period of $T[i..i+2\ell)$.
- If $c_{\min} \neq \infty$ then $c_1 = j + c_{\min} - 1$. Otherwise, $c_0$ does not exist.
- If $c_{\max} \neq -\infty$ then $c_h = j + c_{\max} - 1$. Otherwise, $c_h$ does not exist.
- If $y_{\max} \neq -\infty$ then $j''_{\max} = j + y_{\max} - 1$. Otherwise, $j''_{\max} = j - 1$.

Hence we can compute $j'_{\max}$ in constant time as described above, and output the query result $j_{\max} = \min(j'_{\max}, j''_{\max})$ in constant time. ◀

## References


1. Maxim Babenko, Pawel Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *Proceedings of the 26th Annual Symposium on Discrete Algorithms (SODA 2015)*, pages 572–591, San Diego, CA, USA, January 2015. doi:10.1137/1.9781611973730.39.
2. Golnaz Badkobeh, Maxime Crochemore, Jonas Ellert, and Cyril Nicaud. Back-to-front online Lyndon forest construction. In *Proceedings of the 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)*, pages 13:1–13:23, Prague, Czech Republic, June 2022. doi:10.4230/LIPIcs.CPM.2022.13.
3. Uwe Baier. Linear-time Suffix Sorting – A New Approach for Suffix Array Construction. In *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, pages 23:1–23:12, Tel Aviv, Israel, June 2016. doi:10.4230/LIPIcs.CPM.2016.23.
4. Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
5. Nico Bertram, Jonas Ellert, and Johannes Fischer. Lyndon words accelerate suffix sorting. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA 2021)*, pages 15:1–15:13, Lisbon, Portugal (Virtual Conference), September 2021. doi:10.4230/LIPIcs.ESA.2021.15.
6. Philip Bille, Jonas Ellert, Johannes Fischer, Inge Li Gørtz, Florian Kurpicz, J. Ian Munro, and Eva Rotenberg. Space efficient construction of Lyndon arrays in linear time. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, pages 14:1–14:18, Saarbrücken, Germany (Virtual Conference), July 2020. doi:10.4230/LIPIcs.ICALP.2020.14.
7. P. Bonizzoni, M. Costantini, C. De Felice, A. Petescia, Y. Pirola, M. Previtali, R. Rizzi, J. Stoye, R. Zaccagnino, and R. Zizza. Numeric lyndon-based feature embedding of sequencing reads for machine learning approaches. *Information Sciences*, 607:458–476, 2022. doi:10.1016/j.ins.2022.06.005.

8 Paola Bonizzoni, Alessia Petescia, Yuri Pirola, Raffaella Rizzi, Rocco Zaccagnino, and Rosalba Zizza. Kfinger: Capturing overlaps between long reads by using lyndon fingerprints. In *Proceedings of the International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO 2022)*, pages 436–449, Gran Canaria, Spain, June 2022. doi:10.1007/978-3-031-07802-6_37.

9 K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus, iv. the quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958. doi:10.2307/1970044.

10 Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 74(5):796–807, 2008. doi:10.1016/j.jcss.2007.09.003.

11 Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. *Theoretical Computer Science*, 412(27):2931–2941, 2011. doi:10.1016/j.tcs.2010.06.019.

12 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Ritu Kundu, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Near-optimal computation of runs over general alphabet via non-crossing LCE queries. In *Proceedings of the 23rd International Symposium on String Processing and Information Retrieval (SPIRE 2016)*, pages 22–34, Beppu, Japan, October 2016. doi:10.1007/978-3-319-46049-9_3.

13 Maxime Crochemore and Luís M. S. Russo. Cartesian and Lyndon trees. *Theoretical Computer Science*, 806:1–9, 2020. doi:10.1016/j.tcs.2018.08.011.

14 Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing base without losing space. In *Proceedings of the 42nd Symposium on Theory of Computing (STOC 2010)*, pages 593–602, Cambridge, MA, USA, June 2010. doi:10.1145/1806689.1806771.

15 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. doi:10.1016/0196-6774(83)90017-2.

16 Jonas Ellert. Lyndon Arrays Simplified. In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA 2022)*, pages 48:1–48:14, Potsdam, Germany, September 2022. doi:10.4230/LIPIcs.ESA.2022.48.

17 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, pages 63:1–63:16, Glasgow, Scotland (Virtual Conference), July 2021. doi:10.4230/LIPIcs.ICALP.2021.63.

18 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

19 Johannes Fischer. Optimal succinctness for range minimum queries. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics (LATIN 2010)*, pages 158–169, Oaxaca, Mexico, April 2010. doi:10.1007/978-3-642-12200-2_16.

20 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM 2006)*, pages 36–48, Barcelona, Spain, July 2006. doi:10.1007/11780441_5.

21 Frantisek Franek, A. S. M. Sohidull Islam, Mohammad Sohel Rahman, and William F. Smyth. Algorithms to compute the Lyndon array. In *Proceedings of the Prague Stringology Conference (PSC 2016)*, pages 172–184, Prague, Czech Republic, August 2016. URL: http://www.stringology.org/event/2016/p15.html.

22 Pawel Gawrychowski, Tomasz Kociumaka, Wojciech Rytter, and Tomasz Walen. Faster longest common extension queries in strings over general alphabets. In *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, pages 5:1–5:13, Tel Aviv, Israel, June 2016. doi:10.4230/LIPIcs.CPM.2016.5.

23 Torben Hagerup. Sorting and searching on the word ram. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998)*, pages 366–398, Paris, France, February 1998. doi:10.1007/BFb0028575.

24 Christophe Hohlweg and Christophe Reutenauer. Lyndon words, permutations and trees. *Theoretical Computer Science*, 307(1):173–178, 2003. doi:10.1016/S0304-3975(03)00099-9.

25 Yusaku Kaneta. Fast wavelet tree construction in practice. In *Proceedings of the 25th International Symposium on String Processing and Information Retrieval (SPIRE 2018)*, pages 218–232, Lima, Peru, October 2018. doi:10.1007/978-3-030-00479-8_18.

26 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of the 51st Annual Symposium on Theory of Computing (STOC 2019)*, pages 756–767, Phoenix, AZ, USA, June 2019. doi:10.1145/3313276.3316368.

27 Dominik Kempa and Tomasz Kociumaka. Breaking the o(n)-barrier in the construction of compressed suffix arrays and suffix trees. In *Proceedings of the 34th Annual Symposium on Discrete Algorithms (SODA)*, pages 5122–5202, Florence, Italy, January 2023. doi:10.1137/1.9781611977554.ch187.

28 Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: a unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.

29 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 596–604, New York, NY, USA, October 1999. doi:10.1109/SFFCS.1999.814634.

30 Dmitry Kosolobov. Computing runs on a general alphabet. *Information Processing Letters*, 116(3):241–244, 2016. doi:10.1016/j.ipl.2015.11.016.

31 Felipe A. Louza, Sabrina Mantaci, Giovanni Manzini, Marinella Sciortino, and Guilherme P. Telles. Inducing the Lyndon array. In *Proceedings of the 26th International Symposium on String Processing and Information Retrieval (SPIRE 2019)*, pages 138–151, Segovia, Spain, October 2019. doi:10.1007/978-3-030-32686-9_10.

32 R. C. Lyndon. On Burnside problem I. *Transactions of the American Mathematical Society*, 77(2):202–215, 1954. doi:10.2307/1990868.

33 Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the 1st Annual Symposium on Discrete Algorithms (SODA 1990)*, pages 319–327, San Francisco, CA, USA, January 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320218.

34 Masamichi Miyazaki, Ayumi Shinohara, and Masayuki Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM 1997)*, pages 1–11, Aarhus, Denmark, June 1997. doi:10.1007/3-540-63220-4_45.

35 J. Ian Munro, Yakov Nekrich, and Jeffrey S. Vitter. Fast construction of wavelet trees. In *Proceedings of the 21st International Symposium on String Processing and Information Retrieval (SPIRE 2014)*, Ouro Preto, Brazil, October 2014. doi:10.1007/978-3-319-11918-2_10.

36 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

37 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 10(3), May 2014. doi:10.1145/2601073.

38 Jannik Olbrich, Enno Ohlebusch, and Thomas Büchler. On the optimisation of the GSACA suffix array construction algorithm. In *Proceedings of the 29th International Symposium on String Processing and Information Retrieval (SPIRE 2022)*, pages 99–113, Concepción, Chile, November 2022. doi:10.1007/978-3-031-20643-6_8.

39 Simon J. Puglisi, Jamie Simpson, and W.F. Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401(1):165–171, 2008. doi:10.1016/j.tcs.2008.04.020.

40 Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2006)*, pages 184–195, Marseille, France, February 2006. doi:10.1007/11672142_14.

# Sorting Finite Automata via Partition Refinement

**Ruben Becker**
Ca' Foscari University of Venice, Italy

**Manuel Cáceres**
University of Helsinki, Finland

**Davide Cenzato**
Ca' Foscari University of Venice, Italy

**Sung-Hwan Kim**
Ca' Foscari University of Venice, Italy

**Bojana Kodric**
Ca' Foscari University of Venice, Italy

**Francisco Olivares**
CeBiB – Centre for Biotechnology and Bioengineering, Santiago, Chile
Department of Computer Science, University of Chile, Santiago, Chile

**Nicola Prezza**
Ca' Foscari University of Venice, Italy

## Abstract

Wheeler nondeterministic finite automata (WNFAs) were introduced in (Gagie et al., TCS 2017) as a powerful generalization of prefix sorting from strings to labeled graphs. WNFAs admit optimal solutions to classic hard problems on labeled graphs and languages such as compression and regular expression matching. The problem of deciding whether a given NFA is Wheeler is known to be NP-complete (Gibney and Thankachan, ESA 2019). Recently, however, Alanko et al. (Information and Computation 2021) showed how to side-step this complexity by switching to *preorders*: letting $Q$ be the set of states and $\delta$ the set of transitions, they provided a $O(|\delta| \cdot |Q|^2)$-time algorithm computing a totally-ordered *partition* (i.e. equivalence relation) of the WNFA's states such that (1) equivalent states recognize the same regular language, and (2) the order of (the classes of) non-equivalent states is consistent with any Wheeler order, when one exists. As a result, the output is a preorder of the states as useful for pattern matching as standard Wheeler orders.

Further extensions of this line of work (Cotumaccio et al., SODA 2021 and DCC 2022) generalized these concepts to arbitrary NFAs by introducing *co-lex partial preorders*: in general, any NFA admits a partial preorder of its states reflecting the co-lexicographic order of their accepted strings; the smaller the width of such preorder is, the faster regular expression matching queries can be performed. To date, the fastest algorithm for computing the smallest-width partial preorder on NFAs runs in $O(|\delta|^2 + |Q|^{5/2})$ time (Cotumaccio, DCC 2022), while on DFAs the same task can be accomplished in $O(\min(|Q|^2 \log |Q|, |\delta| \cdot |Q|))$ time (Kim et al., CPM 2023).

In this paper, we provide much more efficient solutions to the co-lex order computation problem. Our results are achieved by extending a classic algorithm for the relational coarsest partition refinement problem of Paige and Tarjan to work with *ordered* partitions. More specifically, we provide a $O(|\delta| \log |Q|)$-time algorithm computing a co-lex total preorder when the input is a Wheeler NFA, and an algorithm with the same time complexity computing the smallest-width co-lex partial order of any DFA. In addition, we present implementations of our algorithms and show that they are very efficient also in practice.

**2012 ACM Subject Classification** Theory of computation → Sorting and searching; Theory of computation → Graph algorithms analysis; Theory of computation → Pattern matching

**Keywords and phrases** Wheeler automata, prefix sorting, pattern matching, graph compression, sorting, partition refinement

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.15



31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Related Version** *Full Version*: http://arxiv.org/abs/2305.05129

**Supplementary Material** *Software (Source Code)*:
https://github.com/regindex/finite-automata-partition-refinement

**Funding** *Ruben Becker, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Nicola Prezza*: Funded by the European Union (ERC, REGINDEX, 101039208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
*Manuel Cáceres*: Funded by the Academy of Finland (grants No. 352821, 328877).
*Francisco Olivares*: Funded by Ph.D Scholarship 21210579, ANID, Chile.

## 1 Introduction

The classical *pattern matching* problem between two strings $S$ (the text) and $P$ (the pattern) over alphabet $\Sigma$ asks to find the substrings of $S$ matching $P$. Although many algorithms solving the *on-line* version of the problem exist, in many scenarios it is possible to pre-process $S$ *off-line* into an *index* to speed up subsequent pattern matching queries. As a matter of fact, a very successful line of research dating back to the invention of suffix trees [24] and culminating with the discovery of compressed data structures [20] showed that it is possible to represent $S$ in compact space while speeding up matching queries.

The *indexed pattern matching* problem can be generalized to collections of strings: in this case the pattern must be found as a substring in a string collection $\mathcal{S}$. A natural approach to solve this problem is to concatenate all strings in $\mathcal{S}$ into one string $S$ and re-use the well-optimized techniques for classical pattern matching. Even though there are many successful examples of indexes following this strategy [12, 14, 19], this approach suffers from high space consumption during index construction (the input is very large), and does not scale to a more general (and interesting) scenario: the case where $\mathcal{S}$ contains an *infinite* number of strings. A solution, addressing both issues, is to represent a (potentially infinite) collection of strings using a finite-state automaton with set of states $Q$ and transition function $\delta$. In this new scenario, the goal of pattern matching is to locate walks in the automaton (seen as a labeled graph) spelling the query pattern $P$. In bioinformatics, for example, genomic collections are represented using pangenome graphs: labeled graphs encoding nucleotide variations within the collection [5, 23]. Matching a DNA sequence over a pangenome graph allows one to discover the genetic variation in a population [22].

Unfortunately, both the off-line and on-line pattern matching problems are hard to solve on labeled graphs: Equi et al. [10, 11] showed that, conditioned on OVH [25], it is not possible to design a polynomial-time algorithm to index a labeled graph such that pattern matching queries can be answered in $O(|P| + |\delta|^{\alpha}|P|^{\beta})$ time, for any constants $\alpha < 1$ or $\beta < 1$.

A successful paradigm to cope with this hardness is to study sub-classes of graphs where the problem is easier. Along these lines, Gagie et al. [13] introduced the class of *Wheeler automata*: labeled graphs admitting a total order of their states (a *Wheeler order*) which respects the underlying alphabet's order and it is propagated through pairs of equally-labeled edges. Wheeler orders generalize prefix sorting (the machinery standing at the core of the most successful string indexes) to labeled graphs and as a consequence, an index over a Wheeler automaton supports pattern matching queries in near-optimal $O(|P| \log |\Sigma|)$ time.

Wheeler automata, however, have two important limitations. First, the classes of Wheeler automata and Wheeler languages (accepted by such automata) are quite restricted. For example, Wheeler automata cannot contain proper cycles labeled with a unary string, and

moreover, Wheeler languages are star-free [2]. Second, several natural problems related to Wheeler graphs are computationally hard. For example, the simple fact of deciding if an automaton is Wheeler is NP-complete, even when the automaton is acyclic [15].

Related to the first issue, the work [8] extended state-ordering to arbitrary automata by using the concept of *co-lex partial order*. By switching from total to partial orders, the authors showed that (i) every automaton can be (partially) sorted, and that (ii) the efficiency of pattern matching on the automaton depends on the *width* (maximum size of an antichain) of such partial order. Wheeler automata are the particular case in which there exists a total co-lex order (i.e., of width one), thus enabling near-optimal time pattern matching queries.

A way to circumvent the latter limitation (hardness of computing a Wheeler order) was proposed by Alanko et al. [2] by switching to *total preorders*: the authors showed a $O(|\delta| \cdot |Q|^2)$-time *partition refinement* algorithm, which outputs a totally-ordered partition of $Q$ such that (1) states in the same part recognize the same language, and (2) the order of the partition is consistent with *any* Wheeler order of $Q$. In other words, this ordered partition is a preorder of the states as useful for indexing as Wheeler orders. Recently, a similar solution was proposed by Chao et al. [4] to speed up the computation of Wheeler orders in practice; after a first polynomial-time partition refinement step, their tool runs an exponential-time solver to assign a Wheeler ordering within classes of equivalent states.

These strategies – partial orders and total preorders – were finally merged by Cotumaccio [6]. As in [8], the efficiency of pattern matching depends on the width of such a partial preorder. The author described a polynomial-time algorithm computing a *colex partial preorder* of smallest width over an arbitrary nondeterministic finite-state automaton (NFA) in $O(|\delta|^2 + |Q|^{5/2})$ time. Later, this running time was improved by Kim et al. [17] in the particular case of DFAs with two algorithms running in time $O(|Q|^2 \log |Q|)$ and $O(|\delta| \cdot |Q|)$, respectively, and one algorithm running in near-optimal time $O(|\delta| \log |Q|)$ on acyclic DFAs. Within the same running times, all the above-mentioned algorithms compute also a chain partition of minimum size $p$ of the partial preorder (by Dilworth's theorem [9], $p$ is equal to the order's width); such a chain partition is needed to build the index described in [6]. The index can be built in linear time given as input such a chain partition, uses $(\lceil \log |\Sigma| \rceil + \lceil \log p \rceil + 2) \cdot (1 + o(1))$ bits per edge, and answers pattern matching queries on the regular language accepted by the automaton in $O(p^2 \log(p \cdot |\Sigma|))$ time per pattern's character.

**Our contributions.** We consider two cases of colex orders: (1) total preorders on NFAs, and (2) partial orders of minimum width on DFAs. We provide algorithms running in time $O(|\delta| \log |Q|)$ for both cases, improving the state-of-the-art cubic and quadratic algorithms to near-optimal time.

Our solution to (1) is obtained by extending a classic algorithm for the relational coarsest partition refinement problem of Paige and Tarjan [21] to work with *ordered* partitions. Our algorithm starts from the ordered partition corresponding to the states' incoming labels and, similarly to [21], iteratively refines this partition by enforcing *forward-stability*: for any two parts $B$ (the "splitter") and $D$ (the "split") the image $\delta_a(B)$ of $B$ through the transition function (for any alphabet's character $a$) must either contain $D$ or be disjoint with $D$. If this condition is not satisfied, $D$ is split into three parts: states reached only from $B$, states reached only from $Q \setminus B$, and states reached from both. In addition to Paige and Tarjan's algorithm, we show how to sort these three parts consistently with the current partition's order. In fact, we show that our algorithm finds a total preorder on a class of automata being strictly larger than the Wheeler automata. We dub this class *quasi-Wheeler*, thereby extending the class of automata that can be indexed to support queries in near-optimal time.

▶ **Theorem 1.** *For an NFA $\mathcal{A} = (Q, \delta, s)$, we can compute a total preorder $\preceq$ in $O(|\delta| \log |Q|)$ time such that $\preceq$ corresponds to a Wheeler preorder if and only if $\mathcal{A}$ is quasi-Wheeler.*

Our solution to (2) relies on a further modification of the partition refinement algorithm. More specifically, we show that within the same time complexity, we can prune the automaton's transition function so that the resulting graph is a quasi-forest satisfying the following property: every state $u$ has only one incoming walk, whose label is co-lexicographically smallest (the *infimum*) among the labels of all walks ending at $u$ in the original DFA. Symmetrically, we compute a pruned automaton encoding the co-lexicographically largest strings (the *suprema*). We obtain our second result by plugging in the $O(|Q| \log |Q|)$-time algorithms of Kim et al. [17], which sort the infimum and suprema strings (suffix-doubling)[1] and then compute a minimum chain partition of the smallest-width co-lex order (interval graph coloring). Our result assumes that the DFA is input-consistent (all in-going transitions to a state are labeled with the same letter). This assumption is a common simplification in automata theory and it is not a limitation of our approach as every DFA can be easily transformed into an equivalent input-consistent DFA.

▶ **Theorem 2.** *For an input-consistent DFA $\mathcal{A} = (Q, \delta, s)$, we can compute a minimum chain partition of the smallest-width co-lex order in $O(|\delta| \log |Q|)$ time.*

We have implemented our partition refinement algorithms. We compared the implementation of our algorithm from Theorem 1 with a heuristic implementation from `WGT` [4] (their first polynomial time step) which, similarly to our algorithm, computes an ordered partition being consistent with any Wheeler order (although not the most refined – unlike the output of our algorithm). On random Wheeler NFAs generated with a tool from the same toolbox (`WGT`), our implementation outperforms the heuristic from `WGT` by more than two orders of magnitude. We also experimentally show that our algorithm from Theorem 2 can prune a pangenomic DFA with more than 50 million states in less than 6 minutes.

## 2 Notation and Preliminaries

For an integer $k \geq 1$, we let $[k] := \{1, \ldots, k\}$. Given a set $U$, a partition $\mathcal{P}$ of $U$ is a set of pairwise disjoint non-empty sets $\{U_1, \ldots, U_k\}$ whose union is $U$. We call $U_1, \ldots, U_k$ the *parts* of $\mathcal{P}$. If we, in addition, have a (total) order of the parts, we say that it is an *ordered partition* and denote it with $\langle U_1, \ldots, U_k \rangle$. For two (ordered) partitions $\mathcal{P}$ and $\mathcal{P}'$ of $U$, we say that $\mathcal{P}'$ is a *refinement* of $\mathcal{P}$ if every part of $\mathcal{P}'$ is contained in a part of $\mathcal{P}$, i.e., for every $U' \in \mathcal{P}'$ there is $U \in \mathcal{P}$ with $U' \subseteq U$. As a special case, a partition is a refinement of itself.

**Relations and Orders.** A *relation* $R$ over a set $U$ is a set of ordered pairs of elements from $U$, i.e., $R \subseteq U \times U$. We sometimes omit $U$ if it is clear from the context. For two elements $u, v$ from $U$, we usually write $uRv$ for $(u, v) \in R$.

A *strict partial order* over a set $U$ is a relation that satisfies irreflexivity, asymmetry, and transitivity. If, in addition, a strict partial order satisfies connectedness it is a *strict total order*. A *total preorder* over a set $U$ is a relation that satisfies transitivity, reflexivity, and connectedness (i.e., for all two distinct $u, v \in U$, $u$ is in relation with $v$ or $v$ is in relation with $u$). An *equivalence relation* over a set $U$ is a relation that satisfies reflexivity, symmetry,

[1] More precisely, it is a special case of the suffix doubling algorithm [17], which runs in the claimed time.

and transitivity. For an equivalence relation $\sim$, we use $[u]_\sim$ to denote the equivalence class of an element $u \in U$ with respect to $\sim$, i.e., $[u]_\sim := \{v \in U : u \sim v\}$. We denote with $U/_\sim$ the partition of $U$ consisting of all equivalence classes $[u]_\sim$ for $u \in U$. In this paper, we denote strict total orders with the symbols $\prec$ and $<$, total preorders with the symbols $\preceq$ and $\leq$, and equivalence relations with the symbols $\sim$ and $\approx$.

A total preorder $\preceq$ over $U$ induces an equivalence relation $\sim$ over $U$: For $u, v \in U$, define $u \sim v$ if and only if $u \preceq v$ and $v \preceq u$. Throughout the paper, $\sim$ will always refer to the equivalence induced by $\preceq$ (the order $\preceq$ will always be unambiguously defined). A total preorder $\preceq$, in addition, yields a strict total order $\prec$ on the elements of $U/_\sim$ as $[u]_\sim \prec [v]_\sim$ if and only if $u \preceq v$ and not $v \preceq u$. Throughout the paper $\prec$ will refer to the strict order induced by $\preceq$ (always unambiguously defined). A total preorder $\preceq$ over a set $U$ can thus be represented by a unique ordered partition $\langle U_1, \ldots, U_k \rangle$, where the parts $U_i$ represent the equivalence classes with respect to $\sim$ and their ordering represents the strict total order $\prec$.

**Non-Deterministic Finite Automata (NFAs).** Let $\Sigma$ denote a fixed finite and non-empty *alphabet* of *letters*. We assume that there is a strict total order $<$ on the alphabet $\Sigma$.

▶ **Definition 3** (NFA and DFA). *A* non-deterministic finite automaton (NFA) *over the alphabet $\Sigma$ is an ordered triple $\mathcal{A} = (Q, \delta, s)$, where $Q$ is the set of* states*, $\delta : Q \times \Sigma \to 2^Q$ is the* transition function*, and $s \in Q$ is the* source state*.*

*A* deterministic finite automaton (DFA) *over the alphabet $\Sigma$ is an NFA over $\Sigma$ such that $|\delta(v, a)| \leq 1$ for all $a \in \Sigma$ and $v \in V$.*

We note that the standard definition of NFAs includes also a set of final states. As we are not concerned with the accepting languages of automata in this work, we omit the final states from the definition. In what follows we consider the alphabet $\Sigma$ to be fixed and thus frequently refer to NFAs without specifying the alphabet. Given an NFA $\mathcal{A} = (Q, \delta, s)$, for a state $u \in Q$ and a letter $a \in \Sigma$, we use the shortcut $\delta_a(u)$ for $\delta(u, a)$, similarly $\delta_a^{-1}(u) = \{v \ : \ \delta(v, a) = u\}$. For a set $S \subseteq Q$, we let $\delta_a(S) := \bigcup_{u \in S} \delta_a(u)$ and $\delta_a^{-1}(S) := \bigcup_{u \in S} \delta_a^{-1}(u)$.

The set of finite strings over $\Sigma$, denoted by $\Sigma^*$, is the set of finite sequences of letters from $\Sigma$. We extend the transition function from letters to finite strings in the common way, i.e, for $\alpha \in \Sigma^*$ and $v \in Q$, we define $\delta_\alpha(v)$ recursively as follows. If $\alpha = \varepsilon$, we let $\delta_\alpha(v) = \{v\}$. Otherwise, if $\alpha = a\alpha'$ for some $a \in \Sigma$ and $\alpha' \in \Sigma^*$, we let $\delta_\alpha(v) = \bigcup_{w \in \delta_a(v)} \delta_{\alpha'}(w)$.

▶ **Definition 4** (Strings Reaching a State). *Given an NFA $\mathcal{A} = (Q, \delta, s)$, for a state $v \in Q$, the* set of strings reaching $v$ *is defined as $S_v = \{\alpha \in \Sigma^* : v \in \delta_\alpha(s)\}$.*

Equivalently, $S_v$ is the regular language recognized by state $v$. For simplicity, in this work we assume that NFAs satisfy the following two properties: (1) The source state has no in-going transitions, i.e., $S_s = \{\varepsilon\}$ and every other state is reachable from the source state, i.e., $|S_v| \geq 1$ for all $v \in Q \setminus \{s\}$. (2) The automaton is *input-consistent*, i.e., for every state $v \in Q \setminus \{s\}$ it holds that $\delta_a^{-1}(v)$ is non-empty for exactly one letter $a \in \Sigma$. It is easy to see that any automaton can be transformed into an equivalent one (in the sense of the recognized language) satisfying assumptions (1) and (2). Condition (1) yields that $|Q| \leq |\delta| + 1$. Condition (2) comes at the cost of increasing the number of states and transitions by a factor of at most $|\Sigma|$. We also assume that a given NFA contains at least one transition for every letter. We denote by $\lambda(v)$ the unique label of the in-going transitions of a state $v \in Q \setminus \{s\}$, while $\lambda(s) := \varepsilon$. We also extend this notation to sets of states, that is for a set of states $S$, we let $\lambda(S) := \{\lambda(v) : v \in S\}$ and if $\lambda(S) = \{a\}$, we write $\lambda(S) = a$.

**Forward-Stable Partitions.** Alanko et al. consider *forward-stable partitions* [2, Section 4.2].

▶ **Definition 5** (Forward-Stability). *Given an NFA $\mathcal{A} = (Q, \delta, s)$ and two sets of states $S, T \subseteq Q$, we say that $S$ is* forward-stable *with respect to $T$, if, for all $a \in \Sigma$, $S \subseteq \delta_a(T)$ or $S \cap \delta_a(T) = \emptyset$ holds. A partition $\mathcal{P}$ of $\mathcal{A}$'s states is* forward-stable for $\mathcal{A}$*, if, for any two parts $S, T \in \mathcal{P}$, it holds that $S$ is forward-stable with respect to $T$.*

A direct consequence of forward-stability is given in the following lemma, i.e., all states in the same part of a forward-stable partition are reached by the exact same set of strings.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a forward-stable partition for an NFA $\mathcal{A} = (Q, \delta, s)$ and assume that $u, v \in P$ for some part $P \in \mathcal{P}$. Then, $S_u = S_v$.*

This property can be proven easily using the definition of forward-stability, see, e.g., [2, Lemma 4.7]. Furthermore, there is a straightforward relationship between forward-stability and bisimulation, see, e.g., the work of Kanellakis and Smolka [16] or Chapter 7.3 of the book by Katoen and Baier [3]. The *coarsest* forward-stable partition for an NFA $\mathcal{A}$, i.e., the forward-stable partition with fewest parts, is identical to the partition consisting of the equivalence classes with respect to the bisimilarity relation (the unique largest bisimulation) on $\mathcal{A}^{-1}$, the automaton obtained from $\mathcal{A}$ by reversing all its transitions. We include a proof of this fact in full version of this article. This also directly yields that there is a unique forward-stable partition. We note that a reverse statement of Lemma 6 may not necessarily hold even for the coarsest forward-stable partition. More precisely, states in different parts of the coarsest forward-stable partition may have the same set of strings reaching them, see the left automaton in Figure 1. In what follows, for an NFA $\mathcal{A} = (Q, \delta, s)$ and an equivalence relation $\sim$ on $Q$, we write $\mathcal{A}/_\sim$ for the quotient NFA (of $\mathcal{A}$ with respect to $\sim$) obtained by collapsing the equivalence classes into single states, see [2, Definition 8] for a formal definition.

**Wheeler NFAs and Quasi-Wheeler NFAs.** Wheeler NFAs [13] are a special class of NFAs that can be stored compactly and indexed efficiently as they can be endowed with a specific type of strict total order, the so-called *Wheeler order*.

▶ **Definition 7** (Wheeler NFA and Wheeler order). *Let $\mathcal{A} = (Q, \delta, s)$ be an NFA. We say that $\mathcal{A}$ is a* Wheeler NFA*, if there exists a* Wheeler order $\prec$ *of $Q$. A Wheeler order $\prec$ of $Q$ is a strict total order on $Q$ such that the source state precedes all other states, i.e., $s \prec v$ for all $v \in Q \setminus \{s\}$, and, for any pair $v \in \delta_a(u)$ and $v' \in \delta_{a'}(u')$:*

**(1)** *If $a < a'$, then $v \prec v'$.*

**(2)** *If $a = a'$, $u \prec u'$, and $v \neq v'$, then $v \prec v'$.*

Clearly, not every NFA is a Wheeler NFA and, even worse, recognizing if a given NFA is Wheeler is NP-complete (for alphabet size at least 2) as shown by Gibney and Thankachan [15]. We now introduce the following problem, called OrderWheeler: Given an arbitrary NFA $\mathcal{A} = (Q, \delta, s)$ as input, the task is to compute a strict total order $\prec$ of $Q$ with the property that $\prec$ is a Wheeler order if $\mathcal{A}$ is Wheeler. As a result of the NP-completeness of recognizing Wheeler NFAs previously mentioned, also OrderWheeler is NP-hard. This follows as checking whether a given order is indeed a Wheeler order can be done in linear time [1, Lemma 3.1]. This motivates introducing the following relaxed version of Wheeler NFAs.

▶ **Definition 8** (Wheeler preorders and quasi-Wheeler NFAs). *Let $\mathcal{A} = (Q, \delta, s)$ be an NFA. A Wheeler preorder $\preceq$ on $Q$ is a total preorder on $Q$ such that:*

- *The partition $Q/_\sim$, where $\sim$ is the equivalence relation induced by $\preceq$, is equal to the coarsest forward-stable partition of $\mathcal{A}$.*
- *The quotient automaton $\mathcal{A}/_\sim$ is a Wheeler NFA with the strict total order $\prec$ induced by $\preceq$ on the equivalence classes with respect to $\sim$.*

*We say that $\mathcal{A}$ is a* quasi-Wheeler NFA*, if there exists a* Wheeler preorder $\preceq$ *on $Q$.*

See the beginning of this section for the formal definition of $\sim$ and $\prec$. From the point of view of indexing, quasi-Wheeler NFAs are as useful as Wheeler NFAs: note that the quotient automaton $\mathcal{A}/_\sim$ in the definition above generates the same language as the original NFA $\mathcal{A}$ due to the properties of the forward-stable partition, see Lemma 6. However, $\mathcal{A}/_\sim$ is a Wheeler NFA and can thus be stored compactly and indexed efficiently. We next note that Corollary 4.11 in the paper by Alanko et al. [2] directly yields the following lemma.

▶ **Lemma 9.** *Let $\mathcal{A} = (Q, \delta, s)$ be a Wheeler NFA, then $\mathcal{A}$ is also a quasi-Wheeler NFA.*

Furthermore, the set of quasi-Wheeler NFAs is strictly larger than the set of Wheeler NFAs as there exist non-Wheeler NFAs $\mathcal{A}$ for which the quotient automaton $\mathcal{A}/_\sim$ is Wheeler, e.g., the NFA and its corresponding quotient NFA in the center and on the right of Figure 1.

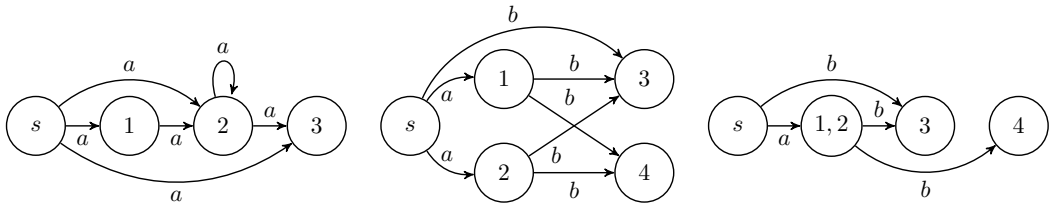



**Figure 1** Left: An NFA for which $\mathcal{P} = \langle\{s\}, \{1\}, \{2\}, \{3\}\rangle$ is the coarsest forward-stable partition, yet $S_2 = S_3$. Center: A non-Wheeler NFA (the two states 3 and 4 cannot be ordered) that is quasi-Wheeler as witnessed by the Wheeler preorder $\preceq$ corresponding to the ordered partition $\langle\{s\}, \{1, 2\}, \{3\}, \{4\}\rangle$. Right: Quotient automaton $\mathcal{A}/_\sim$, where $\mathcal{A}$ is the automaton in the middle and $\sim$ is the equivalence relation induced by the Wheeler preorder $\preceq$. Notice that $\mathcal{A}/_\sim$ is Wheeler with the strict total order $\prec$ induced by $\preceq$ on the equivalence classes with respect to $\sim$.

An important advantage of quasi-Wheeler NFAs over classical Wheeler NFAs is that the former can be recognized in polynomial time, which is implicit in the Forward Algorithm of Alanko et al. [2]. Indeed, the Forward Algorithm receives a Wheeler NFA as input and, in $O(|\delta||Q|^2)$ time, outputs a Wheeler order of the quotient automaton $\mathcal{A}/_\sim$, where $\sim$ is the equivalence relation induced by the returned partition. However, this algorithm actually solves a slightly more general problem, since their output partition is guaranteed to correspond to a Wheeler order of $\mathcal{A}/_\sim$ whenever $\mathcal{A}$ is quasi-Wheeler rather than Wheeler. In other words Alanko et al. gave a polynomial time algorithm for the following problem that we call PreOrderWheeler: Given an arbitrary NFA $\mathcal{A} = (Q, \delta, s)$, the task is to compute a total preorder $\preceq$ of $Q$ with the property that $\preceq$ is a Wheeler preorder if $\mathcal{A}$ is quasi-Wheeler. Their polynomial time algorithm for PreOrderWheeler, the *Forward Algorithm*, yields a polynomial-time recognition algorithm for quasi-Wheeler NFAs as follows: Given an arbitrary input NFA $\mathcal{A}$, run the Forward Algorithm to compute the Wheeler preorder $\preceq$ given by an ordering of the forward-stable partition. The output partition is guaranteed to be the coarsest forward-stable partition. Compute the quotient automaton with respect to this partition and check if it is Wheeler (in polynomial time by [1, Lemma 3.1]) when endowed with the induced strict total order $\prec$.

## 3 Partition Refinement for Wheeler Preorders of NFAs

In this section we provide an algorithm that solves PreOrderWheeler in $O(|\delta| \log |Q|)$ time. Recall that the Forward Algorithm of Alanko et al. [2] has running time $O(|\delta| \cdot |Q|^2)$. We achieve the nearly-linear time complexity by using the partition refinement framework of Paige and Tarjan [21]. It is in fact clear that this framework can be used to compute a forward-stable partition. Our notion of forward-stability corresponds to stability with respect to $|\Sigma|$ relations, defined as $E_a := \{(u, v) \in Q^2 : u \in \delta_a(v)\}$ for $a \in \Sigma$, in their terminology [21, Section 3]. Our main contribution here is to extend the framework so as to compute an *ordered* partition (and thus a preorder), while maintaining the nearly-linear running time.

We proceed with a description of the partition refinement algorithm by Paige and Tarjan. The algorithm maintains two partitions $\mathcal{P}$ and $\mathcal{X}$, here $\mathcal{P}$ is an input partition to be refined and $\mathcal{X}$ is such that (1) $\mathcal{P}$ is a refinement of $\mathcal{X}$ and (2) every part of $\mathcal{P}$ is stable with respect to every part of $\mathcal{X}$. Initially, $\mathcal{X}$ is the partition with a single part containing all elements. Until $\mathcal{P}$ becomes stable, i.e., every part of $\mathcal{P}$ is stable with respect to every part of $\mathcal{P}$, which is witnessed by the fact that $\mathcal{P} = \mathcal{X}$, the algorithm iteratively chooses a *compound part* $S$ from $\mathcal{X}$, i.e., a part form $\mathcal{X}$ that consists of multiple parts in $\mathcal{P}$. Then a "splitter" $B$ is chosen as one of the parts in $\mathcal{P}$ contained in $S$ and every part in $\mathcal{P}$ is refined using $B$ by doing a so-called "three-way split" that we detail in the next section. The idea behind the three-way split is essentially to make $\mathcal{P}$ stable with respect to $B$ and $S \setminus B$ at the same time. In fact, the choice of $B$ in the algorithm of Paige and Tarjan is crucial. It is fundamental to always choose a block $B$ such that $|B| \leq |S|/2$. This is the essential property that yields the nearly linear running time using the observation that every element is contained in at most logarithmically many splitters. This property on the size of $B$ is achieved in Paige and Tarjan's implementation by inspecting the first two elements of the list of all parts from $\mathcal{P}$ contained in $S$ and then choosing $B$ to be the smaller one (in size).

As it turns out, this choice of $B$ however interferes with maintaining the order of the parts such as to satisfy the properties of the Wheeler preorder. We solve this issue by choosing the smaller (in size) among the first and last block (in the sense of the ordered partition) contained in the first compound block (rather than the smaller out of the first two blocks, as in Paige and Tarjan's algorithm). This choice both guarantees that $|B| \leq |S|/2$ and enables us to maintain a consistent ordering between the parts resulting from a split step.

**Algorithm.** We proceed with a description of our algorithm. A pseudo-code formulation can be found in Algorithm 1. First, note that an input-consistent NFA has a natural partition of its states into the $|\Sigma|+1$ parts $\{Q_a\}_{a \in \Sigma \cup \{\varepsilon\}}$, where $Q_a := \{v \in Q : \lambda(v) = a\}$ for $a \in \Sigma \cup \{\varepsilon\}$. Property (1) of Wheeler orders already defines the ordering of these parts: any Wheeler preorder of the NFA's states has to satisfy that $u$ precedes $v$ if $u$'s in-coming letter is smaller than $v$'s. Hence, the ordering of the $|\Sigma|+1$ parts has to be $Q_\varepsilon, Q_{a_1}, \ldots Q_{a_k}$, where we assume that $\Sigma = \{a_1, \ldots, a_k\}$ with $a_1 < \ldots < a_k$. Following this observation, the ordered partition $\mathcal{P}$ in the algorithm is initialized to $\mathcal{P} := \langle Q_\varepsilon, Q_{a_1}, \ldots Q_{a_k} \rangle$.

As in the original partition refinement framework, the algorithm then keeps splitting the parts of the partition using so called *splitters* which are themselves parts in the partition. By splitting, we mean the operation of making all parts of the partition forward-stable with respect to the splitter. As in Paige and Tarjan's algorithm, to maintain the set of splitters, the algorithm also maintains another ordered partition $\mathcal{X}$ such that $\mathcal{P}$ is a refinement of $\mathcal{X}$. Initially $\mathcal{X}$ has a unique part that is equal to $Q$ and the algorithm will maintain the invariant that $\mathcal{P}$ is forward-stable with respect to each part from $\mathcal{X}$. As before, we call the

**Algorithm 1** Ordered Partition Refinement.

**Input** : NFA $\mathcal{A} = (Q, \delta, s)$
**Output** : Ordered Partition $\mathcal{P}$ of $Q$ that corresponds to Wheeler preorder if and only if $\mathcal{A}$ is quasi-Wheeler

```
// * initialization * //
```
$\mathcal{P} := \langle Q_\varepsilon, Q_{a_1}, ..., Q_{a_k} \rangle$, $\mathcal{X} := \langle Q \rangle$
**while** $\mathcal{X} \neq \mathcal{P}$ **do**
```
// * get splitter B * //
```
$S :=$ first block in $\mathcal{X}$ consisting of multiple blocks in $\mathcal{P}$
$B :=$ smaller of first and last block from $\mathcal{P}$ contained in $S$
```
// * variable B contains this block even if split *//
// * update X * //
```
**if** $B$ *is first block* **then** replace $S$ in $\mathcal{X}$ with $B, S \setminus B$ **else** with $S \setminus B, B$
```
// * split blocks in P using B * //
```
**for** $D \in \mathcal{P}$ **do**
```
// * split D * //
```
$D_1 := D \cap \delta_a(B)$, $D_2 := D \setminus D_1$, where $a = \lambda(D)$
$D_{11} := D_1 \cap \delta_a(S \setminus B)$, $D_{12} := D_1 \setminus D_{11}$
```
// * update P * //
```
**if** $B$ *first block* **then** replace $D$ in $\mathcal{P}$ with non-empty sets from $D_{12}, D_{11}, D_2$
**else** with non-empty sets from $D_2, D_{11}, D_{12}$
```
// * continue for behind newly inserted blocks in P * //
```
**return** $\mathcal{P}$

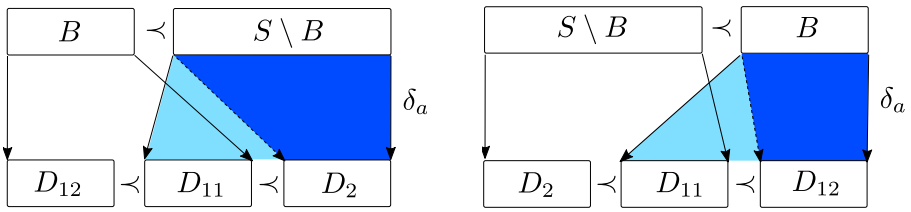



**Figure 2** The two cases of our "ordered" three-way split of $D$ into $D_2$, $D_{11}$ and $D_{12}$ using splitter $B$ contained in the compound part $S$. Here, $D_{11}$ is the part of $D$ that is reached both by $B$ and $S \setminus B$, $D_{12}$ is the part of $D$ reached by $B$ but not by $S \setminus B$, and $D_2$ is the part of $D$ reached by $S \setminus B$ but not by $B$. The size of the splitter $B$ is at most half of the size of the compound block $S$ containing $B$. On the left, $B$ is the first part in $S$ (in the ordered partition), on the right it is the last part. On the left, the order of $B, S \setminus B$ gets propagated forward, the resulting order within $D$ is $D_{12}, D_{11}, D_2$. On the right, the order of $S \setminus B, B$ gets propagated forward, the resulting order within $D$ is $D_2, D_{11}, D_{12}$. The two shades of blue indicate the pruning of edges explained in Section 4.

parts of $\mathcal{X}$ that contain more than a single part from $\mathcal{X}$ *compound parts*. The algorithm then iteratively takes the first (in the order of the ordered partition) compound block $S$ from $\mathcal{P}$ and defines the splitter $B$ as the smaller (in size) of the first and last (in the order of the ordered partition) block in $\mathcal{P}$ contained in $S$.

Once the splitter $B$ is defined, the algorithm aims to split each part $D$ of the partition $\mathcal{P}$ (including $B$ itself) using $B$. As we aim at making the partition forward-stable with respect to $B$ we would want to split $D$ into $D_1 := D \cap \delta_a(B)$ and $D_2 := D \setminus D_1$. To implement the three-way split, we however want to further refine $D$ by $S \setminus B$. Recall that $S$ was a part in $\mathcal{X}$ that contained $B$ and that $\mathcal{P}$ is already forward-stable with respect to $S$ by the invariant. The forward-stability of $D$ with respect to $S$ yields that all states in $D_2$ are reached by $S \setminus B$ and thus $D$ is decomposed into only three parts $D_{11} := D_1 \cap \delta_a(S \setminus B)$, $D_{12} := D_1 \setminus D_{11}$, and $D_2 := D \setminus D_1$ when splitting both with $B$ and $S \setminus B$. This three-way split can be implemented with work proportional to the number of edges leaving $B$ (rather than $B$ and $S \setminus B$). Together with the choice of $B$ being the smaller out of the first and the last block contained in $S$, this is the main property that allows to prove the nearly linear running time bound of the algorithm. The order in which the sets $D_{11}$, $D_{12}$, and $D_2$ are put in $\mathcal{P}$ replacing $D$ depends on whether $B$ was the first or the last block in $S$ and is chosen so as to satisfy property 2 of Wheeler orders. Intuitively, the order of $B$ and $S \setminus B$ is propagated forward to $D_{11}$, $D_{12}$, and $D_2$, see Figure 2 for an illustration of this "ordered" three-way split.

The algorithm keeps splitting the parts in $\mathcal{P}$ in this way until $\mathcal{X} = \mathcal{P}$, i.e., until there are no compound blocks left. Stopping at this point is correct by the invariant that the partition $\mathcal{P}$ is forward-stable with respect to each part from $\mathcal{X} = \mathcal{P}$, and, thus, forward-stable for $\mathcal{A}$.

**Analysis.** The following lemma states that the claimed invariant, that every part of $\mathcal{P}$ is forward-stable with respect to every part of $\mathcal{X}$, holds. This follows immediately from the same property of the framework by Paige and Tarjan (the proof of this lemma can be found in the full version).

▶ **Lemma 10.** *At the beginning of every iteration of the while loop in Algorithm 1, it holds that every part of the ordered partition $\mathcal{P}$ is forward-stable with respect to every part of $\mathcal{X}$.*

We will now prove the core technical result of this section: The ordered partition refinement algorithm in fact computes a partition that corresponds to a Wheeler preorder.

▶ **Lemma 11.** *Assume that Algorithm 1 is called on a quasi-Wheeler NFA $\mathcal{A} = (Q, s, \delta)$. Then, at any step of the algorithm, the partition $\mathcal{P} = \langle Q_1, \dots, Q_k \rangle$ agrees with every Wheeler order $\prec$ of the quotient automaton $\mathcal{A}' := \mathcal{A}/_{\mathcal{P}'}$, where $\mathcal{P}'$ denotes the coarsest forward-stable partition for $\mathcal{A}$. That is, if $i < j, u \in P_i, v \in P_j$ then $u \prec v$ for every Wheeler order $\prec$ of $\mathcal{A}'$.*

**Proof.** The initial partition $\mathcal{P} = \langle Q_\varepsilon, Q_{a_1}, \dots, Q_{a_k} \rangle$ agrees with any Wheeler order $\prec$ of $\mathcal{A}'$. We will show by induction over the number of steps of the while loop that this is the case in any step of the algorithm. Assume that this is true before some intermediate iteration and let us denote with $\mathcal{P}$ the ordered partition at that point. Now, let $S$ be the compound block and let $B$ be the splitter chosen in that iteration. Let us call $\mathcal{P}'$ the ordered partition after refining $\mathcal{P}$ with $B$ and $S \setminus B$, i.e., at the end of that iteration. We will prove that $\mathcal{P}'$ agrees with any Wheeler order $\prec$. As in the algorithm, for $D \in \mathcal{P}$, assume that $a = \lambda(D)$ and let $D_1 = D \cap \delta_a(B)$, $D_2 = D \setminus D_1$, and $D_{11} = D_1 \cap \delta_a(S \setminus B)$ and $D_{12} = D_1 \setminus D_{11}$. Now let $u, v \in D$. If $u$ and $v$ are contained in the same set out of the three sets $D_{12}, D_{11}, D_2$, nothing is to be shown. Hence, assume that $u$ and $v$ are in two different sets out of the three sets. There are three cases: (1) $u \in D_{11}$ and $v \in D_2$, (2) $u \in D_{12}$ and $v \in D_2$, and (3) $u \in D_{12}$ and $v \in D_{11}$. Notice first that in all three cases, there exists $x \in B$ with $u \in \delta_a(x)$. Now, recall that $D$ is stable with respect to $S$ due to Lemma 10 and thus there exists $y \in S \setminus B$ with $v \in \delta_a(y)$ in all three cases as well. Let us now first assume that $B$ is the first block in $S$ in line 5. Then, as $B$ precedes all blocks in $S \setminus B$ in $\mathcal{P}$ and $\mathcal{P}$ is consistent with any Wheeler

order of $\mathcal{A}'$, we have that $x \prec y$ for any Wheeler order of $\mathcal{A}'$. Then, property 2 of Wheeler orders implies that $u \prec v$ for any Wheeler order of $\mathcal{A}'$. In summary, as the algorithm replaces $D$ by $D_{12}, D_{11}, D_2$ in this case, we deduce that $\mathcal{P}'$ again agrees with each Wheeler order of $\mathcal{A}'$. If, instead $B$ is the last block in $S$ in line 5, then $B$ succeeds all blocks in $S \setminus B$ in $\mathcal{P}$ and thus we can analogously deduce $v \prec u$ from property 2 of Wheeler orders and as the algorithm replaces $D$ by $D_2, D_{11}, D_{12}$, we deduce that $\mathcal{P}'$ agrees with every Wheeler order of $\mathcal{A}'$ also in this case. ◀

It remains to argue that the ordered partition refinement algorithm in fact runs in the claimed running time bound of $O(|\delta| \log |Q|)$. The two main ingredients are as follows: (1) We always choose the smaller out of the first and the last part contained in $S$ as $B$ and thus $|B| \leq |S|/2$. As a result every state is in at most logarithmically many splitters. (2) We can implement the algorithm in such a way that the work done in a refinement step with splitter $B$ is proportional to the size of $B$ and the number of out-going transitions from $B$. This can be argued completely analogously as done by Paige and Tarjan. We prove our algorithm running time and describe the main data structure details in the full version. In summary, we obtain the following theorem.

▶ **Theorem 1.** *For an NFA $\mathcal{A} = (Q, \delta, s)$, we can compute a total preorder $\preceq$ in $O(|\delta| \log |Q|)$ time such that $\preceq$ corresponds to a Wheeler preorder if and only if $\mathcal{A}$ is quasi-Wheeler.*

## 4 Partition Refinement for Width-Optimal Co-lex Orders of DFAs

Cotumaccio and Prezza [8] propose a way of sidestepping the fact that there may not be a Wheeler order for a given NFA. They show how to index general NFAs using *partial orders*. For this purpose, they define co-lex orders for NFAs as follows.

▶ **Definition 12** (Co-lex Order). *Let $\mathcal{A} = (Q, \delta, s)$ be an NFA. A* co-lex order *for $\mathcal{A}$ is a strict partial order $\prec$ of $Q$ such that the source state precedes all other states, i.e., $s \prec v$ for all $v \in Q \setminus \{s\}$, and, for any pair $v \in \delta_a(u)$ and $v' \in \delta_{a'}(u')$:*
**(1)** *If $a < a'$, then $v \prec v'$.*
**(2)** *If $a = a'$, $v \prec v'$, and $u \neq u'$, then $u \prec u'$.*

The *width* of a strict partial order $\prec$ is defined as the size of the largest antichain, i.e. set of pairwise *incomparable* states, where two distinct states $u, v \in Q$ are said to be incomparable if neither $u \prec v$ nor $v \prec u$ holds. Finding a smallest-width co-lex order of $\mathcal{A}$ has been of particular interest for constructing efficient indexes for pattern matching on automata [8, 6, 7, 17]. We provide more context on co-lex orders in the full version. For DFAs, the following order is known to be the smallest-width co-lex order [8].

▶ **Definition 13.** *Let $\mathcal{A} = (Q, \delta, s)$ be a DFA. The relation $\prec_{\mathcal{A}}$ over $Q$ is defined as follows. For $u, v \in Q$, $u \prec_{\mathcal{A}} v$ holds if and only if $\alpha < \beta$ for all $\alpha \in S_u$ and $\beta \in S_v$.*

In the above definition, the alphabet order $<$ over $\Sigma$ is extended to the co-lexicographical order on strings. For the purpose of the co-lex order $\prec_{\mathcal{A}}$, a state $v \in Q$ can thus be represented solely using upper and lower bounds on $S_v$. In particular, for a state $v \in Q$, let $\inf S_v$ and $\sup S_v$ be the greatest lower bound (infimum) and the least upper bound (supremum) of $S_v$, respectively, which are possibly left-infinite strings. We refer the reader to the full version for a formal definition of these concepts. As shown by Kim et al. [17], the co-lex order $\prec_{\mathcal{A}}$ can be characterized using the co-lex order of $\mathrm{IS}_{\mathcal{A}} := \{\inf S_v : v \in Q\} \cup \{\sup S_v : v \in Q\}$.

▶ **Lemma 14** ([17, Theorem 10]). *Let $\mathcal{A} = (Q, \delta, s)$ be a DFA. Then, for any two distinct states $u, v \in Q$, $u \prec_{\mathcal{A}} v$ holds if and only if $\sup S_u \leq \inf S_v$.*

We note that, for the purpose of indexing, we also need a minimum chain partition of $\prec_{\mathcal{A}}$, i.e., a minimum-size partition of $Q$ such that the states in each part are totally ordered under $\prec_{\mathcal{A}}$. As shown by Kim et al. [17], such a chain partition can be computed in linear time via a greedy algorithm for the interval graph coloring problem, provided that the set $\mathrm{IS}_{\mathcal{A}}$ (implicitly defining an interval in co-lex order for each state) has been computed and sorted.

In this section, we give an algorithm for co-lex sorting the set $\mathrm{IS}_{\mathcal{A}}$ that runs in nearly-linear time $O(|\delta| \log |Q|)$ on any input-consistent DFA. This is a significant improvement with respect to the best previously-known algorithms that run in $O(|\delta| \cdot |Q|)$ and $O(|Q|^2 \log |Q|)$ time [17].

**Outline.** Given an input DFA $\mathcal{A} = (Q, \delta, s)$, we first compute a pruned automaton $\mathcal{A}^{\inf} = (Q, \delta^{\inf}, s)$ that *encodes* $\{\inf S_v : v \in Q\}$ in a sense that walking back starting at a state $v \in Q$ on the pruned transition $\delta^{\inf}$ yields its infimum string $\inf S_v$. Similarly, we compute a pruned automaton $\mathcal{A}^{\sup} = (Q, \delta^{\sup}, s)$ that encodes the supremum strings. As we will see, computing these sets takes $O(|\delta| \log |Q|)$ time each. On these two pruned automata, we compute the co-lex order of $\mathrm{IS}_{\mathcal{A}}$ in $O(|Q| \log |Q|)$ time in a similar way as done by Kim et al. [17], from which the smallest-width co-lex order of $\mathcal{A}$ (with a minimum chain partition) can be computed in linear time.

**Pruning Algorithm.** The algorithm is identical to Algorithm 1 with the only difference that we insert an additional pruning step, Algorithm 2, before Line 8 of the algorithm. See Figure 2 for an illustration: If a state in $D$ is reached from both $B$ and $S \setminus B$, we only keep the edges from the smaller (in the sense of the ordered partition) of the two. This corresponds to changing $\delta_a$ such that the blue portion of the figure shrinks to the dark blue one.

**Algorithm 2** Pruning Step for Algorithm 1 inserted before Line 8.

```
// * prune A * //
```
**if** *$B$ is the first block* **then** delete transitions from $S \setminus B$ to $D_1$ **else** from $B$ to $D_1$.

We inductively show that Algorithm 2 outputs a pruned automaton encoding the set of infima $I := \{\inf S_v : v \in Q\}$. For an integer $i \geq 1$, let $\delta^{(i)}$ be the pruned transition at the end of the $i$-th iteration of the while loop. Let us analogously denote with $\mathcal{X}^{(i)}$ and $\mathcal{P}^{(i)}$ the state of the ordered partitions $\mathcal{X}$ and $\mathcal{P}$ in the algorithm at the end of the $i$-th iteration. For convenience, we also define $\delta^{(0)} = \delta$, $\mathcal{X}^{(0)} = \langle Q \rangle$, and $\mathcal{P}^{(0)} = \langle Q_\varepsilon, Q_{a_1}, \ldots, Q_{a_k} \rangle$. Lemma 10 and the definition of the pruning step yield the following invariant on forward-stability.

▶ **Observation 15.** *At the end of every iteration $i$ of the while loop in Algorithm 1 when run together with the pruning step, it holds that every part of the ordered partition $\mathcal{P}^{(i)}$ is forward-stable with respect to every part in $\mathcal{X}^{(i)}$ in the automaton $\mathcal{A}^{(i)} = (Q, \delta^{(i)}, s)$.*

Intuitively speaking, the pruning step removes transitions that do not originate from the co-lexicographically smallest part in the sorted partition. Hence, we obtain the following lemma (the proof is deferred to the full version).

▶ **Lemma 16.** *Let $v \in Q$ be a state with $a = \lambda(v)$ and let $A, A' \in \mathcal{X}^{(i)}$ for some $i \geq 0$ be such that $v \in \delta_a^{(i)}(A) \cap \delta_a(A')$. Then either (i) $A = A'$ or (ii) $A$ precedes $A'$ in $\mathcal{X}^{(i)}$.*

Let $\mathcal{P}^*$ and $\delta^*$ be the ordered partition and the pruned transition, respectively, obtained at the end of the algorithm's execution. From Lemma 16, we can see that for every state $v \in Q \setminus \{s\}$, there exists a unique part $A \in \mathcal{P}^*$ such that $v \in \delta^*_{\lambda(v)}(A)$. Combining this with Observation 15, we obtain that a unique string can be obtained by walking backwards through the pruned transitions $\delta^*$. For $u \in Q$, let $\alpha^*_u$ be the longest (or possibly left-infinite) string that can be obtained in this way starting from $u$. Since every transition comes from a co-lex smallest part, we can obtain the following lemma (the proof is included in the full version).

▶ **Lemma 17.** *For every $u \in Q$, $\alpha^*_u = \inf S_u$.*

It is worth noting that some states possibly have more than one in-going transitions after the termination of the algorithm. Nevertheless, Lemma 17 still holds when we choose any of them and remove the others. From this observation, we can assume that every state (except the source state) in the pruned automaton $\mathcal{A}^{\inf}$ has exactly one in-going transition.

Similarly, we can compute the pruned automaton for the set of suprema $\{\sup S_v : v \in Q\}$. The only difference is that we start with the partition $\mathcal{P}'^{(0)} = \langle Q_{a_k}, \ldots, Q_{a_1}, Q_\varepsilon \rangle$ with the reversed order. It is simple to see the greatest string can be computed with this setting.

Regarding the time complexity, notice that the partition refinement algorithm iterates through the same partitions as if it were run on $\mathcal{A}^{\inf}$ in the first place. Additional time is taken for the pruning step for deleting transitions. This work can be done in $O(1)$ time per transition. Once a transition is deleted, it will never be considered in the rest of the execution and hence the additional work amortizes to $O(|\delta|)$. Consequently, the asymptotic time complexity of the partition refinement algorithm with pruning remains $O(|\delta| \log |Q|)$.

**Computing Co-Lex Order of $\mathrm{IS}_{\mathcal{A}}$.** Once the two pruned automata $\mathcal{A}^{\inf}$ and $\mathcal{A}^{\sup}$ are obtained, we can easily compute the co-lex order of $\mathrm{IS}_{\mathcal{A}}$ in $O(|Q| \log |Q|)$ time using the suffix doubling algorithm [17], which extends the well-known prefix-doubling algorithm [18]. Instead of accessing via integer indexes for the doubling procedure, each state keeps a pointer referring to another (possibly the same) state that is $2^k$ hops away along its backward walk. We describe all details of this algorithm in the full version. In summary, we conclude this section with the following theorem.

▶ **Theorem 2.** *For an input-consistent DFA $\mathcal{A} = (Q, \delta, s)$, we can compute a minimum chain partition of the smallest-width co-lex order in $O(|\delta| \log |Q|)$ time.*

## 5 Experimental results

We implemented our partition refinement algorithms in `C++` and made it available at `https://github.com/regindex/finite-automata-partition-refinement.git`. We compared the algorithm from Theorem 1 to the only (to the best of our knowledge) other available tool for computing a sorted partition consistent with any Wheeler order of the input NFA, the renaming heuristic from `WGT` [4]. For a fair comparison, we only run the first part of the `WGT` recognizer and remove the exponential time search that is used to subsequently compute a Wheeler order of the states ending up in equivalence classes.[2] We used the Wheeler graph generator included in `WGT` to generate 7 random input Wheeler NFAs with

[2] As observed empirically, the partition computed by `WGT` is in some cases coarser than the partition computed by our algorithm and is, thus, not necessarily forward-stable.

$|Q| \in \{5^6 \cdot 2^i : i = 0, \ldots, 6\}$ and $|\delta| = 3|Q|$ (we cannot generate much denser graphs since for Wheeler graphs $\delta = O(|\Sigma| \cdot |Q|)$ and here $|\Sigma| = 5$). Our experiments were run on a server with Intel(R) Xeon(R) W-2245 CPU @ 3.90GHz with 8 cores and 128 gigabytes of RAM running Ubuntu 18.04 LTS 64-bit.

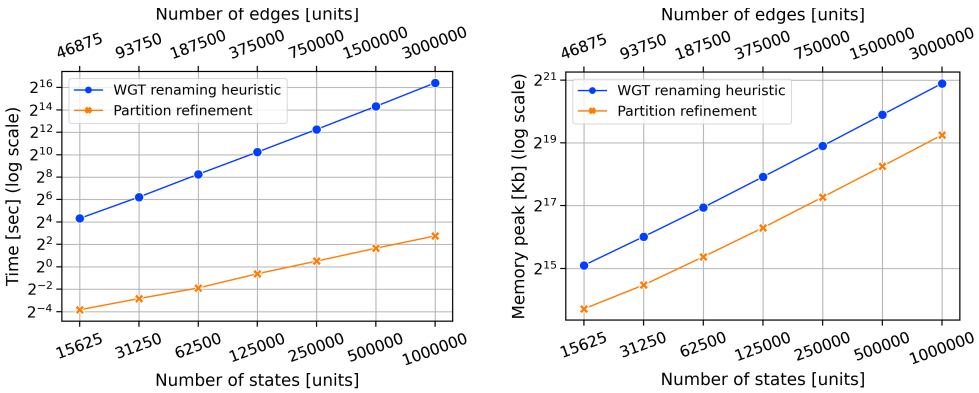



**Figure 3** CPU time (left) and memory peak (right) for sorting seven Wheeler NFAs using our partition refinement algorithm and the renaming heuristic contained in the `WGT` recognizer software. Datasets generated using `WGT` specifying seven different combinations of number of states and edges.

Figure 3 shows the running time and peak memory consumption of both implementations. As expected, our partition refinement implementation shows a slight super-linear behavior confirming the $O(|\delta| \cdot \log |Q|)$ worst-case running time of the algorithm. On the other hand, `WGT` shows a quadratic behavior, which is better than the cubic $O(|\delta| \cdot |Q|^2)$ bound for the Forward Algorithm of Alanko et. al [2], but between $\approx 100\times$ to $\approx 10000\times$ slower than our implementation. In terms of peak memory, both implementations behave linearly with respect to the automaton's size with a $3\times$ advantage in favor of our implementation. On the largest input instance containing one million states, our implementation computes the Wheeler preorder in about seven seconds. In a second experiment, we show that our implementation of the algorithm from Theorem 2 can prune a pangenomic DFA containing over 51 million states and 53 million edges in 355 seconds with a peak memory of 33.2 Gb.

## References


1 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 911–930. SIAM, 2020.

2 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Wheeler languages. *Information and Computation*, 281:104820, 2021. doi:10.1016/j.ic.2021.104820.

3 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

4 Kuan-Hao Chao, Pei-Wei Chen, Sanjit A Seshia, and Ben Langmead. WGT: Tools and algorithms for recognizing, visualizing and generating Wheeler graphs. *bioRxiv*, 2022. doi:10.1101/2022.10.15.512390.

5 Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in bioinformatics*, 19(1):118–135, 2018.

6 Nicola Cotumaccio. Graphs can be succinctly indexed for pattern matching in $O(|E|^2 + |V|^{5/2})$ time. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2022, Snowbird, UT, USA, March 22-25, 2022*, pages 272–281. IEEE, 2022. doi:10.1109/DCC52660.2022.00035.

7 Nicola Cotumaccio, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Co-lexicographically ordering automata and regular languages. Part I, 2022. doi:10.48550/arXiv.2208.04931.

8 Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2585–2599. SIAM, 2021. doi:10.1137/1.9781611976465.153.

9 Robert P. Dilworth. A Decomposition Theorem for Partially Ordered Sets. *Annals of Mathematics*, 51(1):161–166, 1950. doi:10.2307/1969503.

10 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails. In *Proceedings of the 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 608–622, 2021. doi:10.1007/978-3-030-67731-2_44.

11 Massimo Equi, Veli Mäkinen, Alexandru I Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Transactions on Algorithms*, 19(3):1–25, 2023.

12 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. Lz77-based self-indexing with faster pattern matching. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics*, pages 731–742, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

13 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical computer science*, 698:67–78, 2017.

14 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-Time Text Indexing in BWT-runs Bounded Space. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1459–1477, 2018. doi:10.1137/1.9781611975031.96.

15 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

16 Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

17 Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster Prefix-Sorting Algorithms for Deterministic Finite Automata. In *34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023)*, pages 16:1–16:16, 2023. doi:10.4230/LIPIcs.CPM.2023.16.

18 Udi Manber and Gene Myers. Suffix Arrays: A New Method for on-Line String Searches. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 319–327, 1990. doi:10.1137/0222058.

19 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of computational biology : a journal of computational molecular cell biology*, 17:281–308, March 2010. doi:10.1089/cmb.2009.0169.

20 Gonzalo Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016. ISBN 978-1-107-15238-0. 536 pages.

21 Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.

22 Jouni Sirén, Jean Monlong, Xian Chang, Adam M Novak, Jordan M Eizenga, Charles Markello, Jonas A Sibbesen, Glenn Hickey, Pi-Chuan Chang, Andrew Carroll, et al. Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. *Science*, 374(6574):abg8871, 2021.

23 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):375–388, 2014. doi:10.1109/TCBB.2013.2297101.

24 Peter Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE, 1973.

25 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

# Fully Polynomial-Time Algorithms Parameterized by Vertex Integrity Using Fast Matrix Multiplication

**Matthias Bentert**
University of Bergen, Norway

**Klaus Heeger**
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

**Tomohiro Koana**
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

## Abstract

We study the computational complexity of several polynomial-time-solvable graph problems parameterized by *vertex integrity*, a measure of a graph's vulnerability to vertex removal in terms of connectivity. Vertex integrity is the smallest number $\iota$ such that there is a set $S$ of $\iota' \leq \iota$ vertices such that every connected component of $G - S$ contains at most $\iota - \iota'$ vertices. It is known that the vertex integrity lies between the well-studied parameters *vertex cover number* and *tree-depth*. Our work follows similar studies for vertex cover number [Alon and Yuster, ESA 2007] and tree-depth [Iwata, Ogasawara, and Ohsaka, STACS 2018].

Alon and Yuster designed algorithms for graphs with small vertex cover number using fast matrix multiplications. We demonstrate that fast matrix multiplication can also be effectively used when parameterizing by vertex integrity $\iota$ by developing efficient algorithms for problems including an $O(\iota^{\omega-1}n)$-time algorithm for Maximum Matching and an $O(\iota^{(\omega-1)/2}n^2) \subseteq O(\iota^{0.687}n^2)$-time algorithm for All-Pairs Shortest Paths. These algorithms can be faster than previous algorithms parameterized by tree-depth, for which fast matrix multiplication is not known to be effective.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Matchings and factors; Mathematics of computing → Graph algorithms; Theory of computation → Shortest paths

**Keywords and phrases** FPT in P, Algebraic Algorithms, Adaptive Algorithms, Subgraph Detection, Matching, APSP

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.16

**Funding** *Matthias Bentert*: Supported by the European Research Council (ERC) project LOPRE (819416) under the Horizon 2020 research and innovation program.
*Klaus Heeger*: Supported by Deutsche Forschungsgemeinschaft (DFG) project NI 369/16.
*Tomohiro Koana*: Supported by the DFG project DiPa (NI 369/21).

**Acknowledgements** We thank Vincent Borko for fruitful discussions regarding the results for finding small induced subgraphs and anonymous reviewers for their constructive feedback which, in particular, helped improving the running time of our algorithm for All-Pairs Shortest Paths.

## 1 Introduction

Parameterized complexity provides a powerful framework for studying NP-hard problems. The main idea behind parameterized algorithms is to analyze the running time in terms of the input size $|I|$ as well as a parameter $k$, some measure of the input instance. A problem is *fixed-parameter tractable* or *FPT* for short, if it admits an *FPT algorithm*, an algorithm running in time $f(k) \cdot |I|^{O(1)}$ time, where $f$ is a function solely depending on $k$. In the past decade, a line of research dubbed "FPT in P" has emerged, where the goal is a more


refined parameterized analysis of polynomial-time-solvable problems [1, 5, 7, 13, 20, 25, 27]. Although the function $f$ usually has to be at least exponential when working with NP-hard problems, this is not true for problems in $P$. FPT algorithms where $f$ is a polynomial function are called *fully polynomial-time algorithms* and are of course desirable.

We study graph problems in this work. Let $n$ and $m$ be the number of vertices and edges, respectively. Also, let vc be the vertex cover number and td be the tree-depth (see Section 2 for definitions). Alon and Yuster [2] demonstrated that fast matrix multiplication can be used effectively for graphs with a (not necessarily small) vertex cover, developing algorithms for Maximum Matching and All-Pairs Shortest Paths (APSP) that run in $O(n^\omega)$ time (where $\omega < 2.372$ is the matrix multiplication exponent) even when $\mathrm{vc} = \Theta(n)$. More recently, Iwata et al. [26] proposed a divide-and-conquer framework in the design of fully polynomial-time algorithms parameterized by tree-depth. For instance, they showed that Maximum Matching can be solved in $O(m \cdot \mathrm{td})$ time.

In this work, we consider the parameter *vertex integrity*, a parameter that lies between vertex cover number and tree-depth. The vertex integrity $\iota$ of a graph $G$ is the smallest integer such that $G$ contains a set $S$ of $\iota' \leq \iota$ vertices whose deletion results in a graph whose connected components each have at most $\iota - \iota'$ vertices. Many problems can be solved in $O(nm)$ time and thus in $O(\iota n^2)$ time, since $m \in O(\iota n)$. As the relation $\mathrm{td} \leq \iota \leq \mathrm{vc}+1$ holds for any graph, an algorithm that runs in $O(m \cdot \mathrm{td})$ time (e.g., for Maximum Matching) also runs in time $O(\iota^2 n)$. These bounds become $O(n^3)$ when $\iota = \Theta(n)$. However, many problems can be solved in faster $O(n^\omega)$ time using fast matrix multiplication. We aim to close this gap by developing fully polynomial-time algorithms that run in $O(n^\omega)$ time even when $\iota = \Theta(n)$. Such algorithms are called *adaptive*, and are optimal unless there is an (unparameterized) algorithm that runs faster than $O(n^\omega)$ time. For many problems, the discovery of such an algorithm would be a breakthrough, given that these $O(n^\omega)$-time algorithms were developed decades ago and have not been improved since.

**Our approach.** Before describing our results, let us briefly discuss our approach (see Section 2 for details). Let $S$ be a *$k$-separator*, a vertex set of size at most $k' \leq k$ such that each connected component of $G - S$ has size $k - k'$. Throughout the paper, we will make the assumption that a $k$-separator is given as input. We remark that our algorithms do not require an (optimal) $\iota$-separator to compute the correct solution. However, the running times of our algorithm will depend on $k$ and to achieve the claimed running times, we require a $k$-separator with $k \in O(\iota)$. Although $G - S$ may have $O(n)$ connected components, we may assume that there are $\Theta(n/k)$ "components" (which are not necessarily connected; see Section 2 for details), each with $O(k)$ vertices. For every component $C$, we can use the $O(n^\omega)$-time algorithm to solve the instance on $G[C]$ or $G[S \cup C]$, which takes $O(k^\omega \cdot n/k) = O(k^{\omega-1} n)$ time. The next step is to combine solutions for $O(n/k)$ instances, which varies depending on the problem. For instance, this is trivial for the problem of finding a triangle, as a triangle must be contained in $S \cup C$ for some component $C$. For other problems, e.g., finding a maximum matching, this step requires a more sophisticated approach.

**Main results.** There are three main results in this work.

The first result concerns the problem of finding an induced copy of a graph $H$. Vassilevska Williams et al. [38] gave an $O(n^\omega)$-time algorithm that finds an induced copy of $H$ when $H$ is a graph on four vertices that is not a clique $K_4$ or its complement $\overline{K_4}$. Their randomized algorithm is based on computing the number of induced copies of $H$ modulo some integer $q$ which they show to be computable from $A^2$ in linear time, where $A$ is the adjacency matrix. We observe that the "essential" part of $A^2$ can be computed in $O(\iota^{\omega-1} n)$ time, leading to $O(\iota^{\omega-1} n)$-time algorithms.

Secondly, we develop an $O(\iota^{\omega-1}n)$-time algorithm for finding a maximum matching. We start by showing that whether a graph contains a perfect matching can be determined in $O(\iota^{\omega-1}n)$ time. Tutte [36] observed that the Tutte matrix is nonsingular if and only if the graph has a perfect matching. By the Schwartz-Zippel lemma, we can test its nonsingularity in randomized $O(n^\omega)$ time. We can thus test whether each component in $G-S$ has a perfect matching in $O(\iota^{\omega-1}n)$ time. However, there might be a vertex in $S$ that must be matched to a vertex in $G-S$. To handle these cases, we use *Schur complements*. The task of finding a maximum matching is more intricate. Lovász [30] generalized Tutte's observation by stating that the rank of the Tutte matrix (which can be computed in randomized $O(n^\omega)$ time) equals twice the size of a maximum matching. It was only decades later that $O(n^\omega)$-time algorithms for finding one were discovered. Mucha and Sankowski [31] and Harvey [24] gave such algorithms. We show how to adapt the latter to obtain an $O(\iota^{\omega-1}n)$-time algorithm for finding a maximum matching.

Lastly, we study APSP on unweighted graphs. Seidel [34] showed that APSP can be solved in $O(n^\omega \log n)$ time. Alon and Yuster [2] later developed an algorithm that runs in $O(\mathrm{vc}^{\omega-2}\, n^2)$ time (they actually provide a stronger bound using rectangular matrix multiplication). We show that APSP can be solved in $O(\iota^{\omega-2}n^2)$ time when the graph has constant diameter. We were not able to obtain an adaptive algorithm in general, but we give an $O(\iota^{(\omega-1)/2}n^2) \subseteq O(\iota^{0.687}n^2)$-time algorithm. When parameterizing by vc, we can effectively replace every vertex not in the vertex cover with edges of weight two connecting their neighbors. Thus, the $O(Wn^\omega)$-time algorithm [17, 35] for weighted APSP, where $W$ is the maximum weight, finds all pairwise distances between vertices in the vertex cover in $O(\mathrm{vc}^\omega)$ time. For vertex integrity, we show how to replace every component with edges of weight $O(\iota)$. To compute distances between pairs of vertices with at least one vertex not in the $k$-separator, we use the known subcubic-time algorithm for computing min-plus matrix multiplication for *bounded-difference matrices*, matrices in which the difference of two adjacent entries in a row is constant [10].

**Previous work on vertex integrity.** The notion of vertex integrity was introduced by Barefoot et al. [3]. The vertex integrity $\iota$ can be much smaller than $n$, e.g., it is known that $\iota \in O(n^{2/3})$ on $K_h$-minor free graphs [4]. The Vertex Integrity problem, i.e., computing an $\iota$-separator is NP-hard. A straightforward branching algorithm solves Vertex Integrity in $O(\iota^\iota \cdot n)$ time (see [15]). A greedy algorithm can find an $O(\iota^2)$-separator in linear time. There is also a polynomial-time algorithm that can find an $O(\iota \log \iota)$-separator [29]. FPT algorithms parameterized by vertex integrity gained increased attention recently [6, 15, 16, 22, 28]. In particular, see Gima et al. [22] for an extensive list of problems that are W[1]-hard for tree-depth but become FPT when parameterized by vertex integrity.

## 2 Preliminaries

We use standard notation from graph theory. Unless stated otherwise, all appearing graphs are undirected. Further, $V$ denotes the set of vertices in the graph, $E$ its set of edges, $n$ its number of vertices, and $m$ its number of edges. We denote an edge between two vertices $u$ and $v$ by $uv$. A *walk* of length $\ell$ is a sequence $v_1, \ldots, v_\ell$ of (not necessarily distinct) vertices such that $v_iv_{i+1} \in E$ for all $i \in [\ell-1]$, where $[j] := \{1, \ldots, j\}$ for any integer $j$. A walk whose vertices are all pairwise distinct is a *path*. The *adjacency matrix* of $G$ is the $V \times V$-matrix $A$ with $A[u,v] = 1$ if and only if $uv \in E$, and $A[u,v] = 0$ otherwise (where $A[u,v]$ is the entry of $A$ indexed by $u$ and $v$).

**Graph parameters.** For a graph $G$, the *vertex integrity* is the smallest integer $\iota$ such that $G$ contains a set $S$ (called *$\iota$-separator*) of $\iota' \leq \iota$ vertices whose deletion results in a graph whose connected components each have at most $\iota - \iota'$ vertices. The *vertex cover number* vc is the smallest cardinality of a vertex cover, a set that contains at least one endpoint of every edge. The *tree-depth* td is the smallest depth of a rooted forest $F$ with vertex set $V$ such that $G$ can be *embedded* in $F$, i.e., for every edge $xy$ in $G$, $x$ is an ancestor of $y$ or vice versa. The *feedback vertex number* is the smallest cardinality of a feedback vertex set, a set that contains at least one vertex of every cycle.

**Decomposition.** Here, we describe the decomposition with respect to a $k$-separator, which will be used throughout the paper. Let $S$ be a $k$-separator. Typically in our algorithms, we spend $O(k^\omega)$ time for every connected component in $G - S$. Since $G - S$ may have $\Omega(n)$ connected components, this would result in a running time of $O(k^\omega n)$, which is often worse than a more straightforward algorithm. Thus, we will do the following to bound the number of "components" by $O(n/k)$: Basically, we put together some connected components $C$ and construct a collection $\mathcal{T}$ of sets, each (except for possibly the last one) containing between $k$ and $2k-1$ vertices. More precisely, we start with $\mathcal{T} = \emptyset$ and process the connected components of $G - S$ one by one as follows. If every set $T \in \mathcal{T}$ has at least $k$ vertices, then add $\{C\}$ to $\mathcal{T}$, and otherwise replace the set $T \in \mathcal{T}$ with $|T| < k$ by $T \cup C$. Since every connected component $C$ has at most $k$ vertices, every set $T \in \mathcal{T}$ (except for possibly the last set which may be smaller) contains between $k$ and $2k-1$ vertices. Let $\mathcal{T} = \{T_1, \ldots, T_\nu\}$. It is easy to see that $\nu \leq n/k + 1$. In our algorithms, we will always assume that the decomposition $(S; T_1, \ldots, T_\nu)$ of $V$ is given. Note that given a $k$-separator, the decomposition can be computed in linear time.

**Basic operations in matrix multiplication time.** For $n \times n$-matrices $A, B$, one can compute the following in $O(n^\omega)$ time: (i) the product $AB$, (ii) the determinant $\det A$, (iii) the inverse $A^{-1}$, and (iv) a row/column basis of $A$ (see e.g., [9]). More generally, for a $k \times n$-matrix $A$ and an $n \times k$-matrix $B$, one can compute the product $AB$ in $O(k^{\omega-1}n)$ time by dividing $A$ and $B$ into $n/k$ blocks of size $k \times k$. The rank of $A$ can be computed using Gaussian elimination using $O(k^{\omega-1}n)$ arithmetic operations [8]. Throughout the paper, we will use a word RAM model with word size $O(\log n)$. If $\mathbb{F}$ is a field of size $\mathrm{poly}(n)$, we will assume that addition and multiplication take $O(1)$ time.

**Matrices and Matchings.** For a subset $X$ of rows and a subset $Y$ of columns, we denote by $A[X, Y]$ the restriction of the matrix $A$ to rows $X$ and columns $Y$. For a set $X$ of rows (or columns), we will use the shorthand $A[X]$ for $A[X, X]$. The $i$-th power of the adjacency matrix $A$ correspond to the number of walks of length $i$, i.e., $A^i[u, v]$ equals the number of $u$-$v$-walks of length $i$ in $G$.

Note that for any graph with a $k$-separator $S$ and decomposition $(S; T_1, \ldots, T_\nu)$ it holds that there is no edge between a vertex in $T_i$ and a vertex in $T_j$ for any $i \neq j$. We can therefore represent the adjacency matrix $A$ of the graph as follows.

$$A = \begin{array}{c} \\ S \\ \bar{S} \end{array} \begin{array}{c} \begin{array}{cc} S & \bar{S} \end{array} \\ \begin{bmatrix} \gamma & \beta \\ \beta^T & \alpha \end{bmatrix} \end{array}, \text{ for } \alpha = \begin{array}{c} \\ T_1 \\ \vdots \\ T_\nu \end{array} \begin{array}{c} \begin{array}{ccc} T_1 & \cdots & T_\nu \end{array} \\ \begin{bmatrix} \alpha_1 & \cdots & O \\ \vdots & \ddots & \vdots \\ O & \cdots & \alpha_\nu \end{bmatrix} \end{array} \text{ with } \beta = \begin{array}{c} \\ S \end{array} \begin{array}{c} \begin{array}{ccc} T_1 & \cdots & T_\nu \end{array} \\ \begin{bmatrix} \beta_1 & \cdots & \beta_\nu \end{bmatrix} \end{array}. \quad (1)$$

Here, $\bar{S} = T_1 \cup \cdots \cup T_\nu$ and the matrices $\alpha_i, \beta_i, \gamma$ all have size $O(k) \times O(k)$. Many of our algorithms will use this representation and exploit the sparseness when computing e.g., matrix multiplications and determinants.

We denote the (unique) finite field with $2^q$ many elements by $\mathrm{GF}(2^q)$. Note that this field has characteristic 2, i.e., $x + x = 0$ for every $x \in \mathrm{GF}(2^q)$. For a graph $G = (V, E)$, the Tutte matrix (also known as the skew adjacency matrix) $A$ whose rows and columns are indexed by $V = \{v_1, \ldots, v_n\}$ is defined by

$$A[u,v] = \begin{cases} +x_{uv} & \text{if } u = v_i, v = v_j \text{ with } i < j \text{ and } uv \in E(G) \\ -x_{uv} & \text{if } u = v_i, v = v_j \text{ with } j < i \text{ and } uv \in E(G) \\ 0 & \text{otherwise,} \end{cases}$$

where $x_{uv}$ is a variable associated with the edge $uv$. The Tutte matrix $A$ is *skew-symmetric*, i.e., $A = -A^T$. The Pfaffian of a skew-symmetric matrix $A$ indexed by $V$ is defined as

$$\mathrm{pf}(A) = \sum_{M \in \mathcal{M}} \sigma_M \prod_{uv \in M} A[u,v],$$

where $\mathcal{M}$ is the set of all perfect matchings of $(V, \binom{V}{2})$ and $\sigma_M \in \{\pm 1\}$ is the sign of $M$. We will assume that the field has characteristic 2 (implying $-1 = 1$), and thus the precise definition of $\sigma_M$ is not important for us. (This assumption is not essential to our algorithm but it will simplify the notation.)

The following are well-known facts about skew-symmetric matrices (see e.g., [23, 32]).

▶ **Lemma 1.** *For a skew-symmetric matrix $A$, we have $\det A = \mathrm{pf}(A)^2$.*

In particular, a skew-symmetric matrix $A$ is nonsingular if and only if $\mathrm{pf}(A) \neq 0$.

▶ **Lemma 2.** *For a skew-symmetric matrix $A$, if $X$ is a row (or column) basis, then $A[X]$ is nonsingular.*

The next is immediate from the definition of Pfaffians.

▶ **Lemma 3** (row expansion). *For a skew-symmetric matrix $A$ indexed by $V$ and $v \in V$ over a field of characteristic 2, we have $\mathrm{pf}(A) = \sum_{v' \in V \setminus \{v\}} A[v,v'] \cdot \mathrm{pf}(\widehat{A}_{v,v'})$, where $\widehat{A}_{v,v'}$ is the matrix where the rows and columns indexed by $v$ and $v'$ are deleted.*

Proofs of statements marked with $\star$ are omitted from the conference version and can be found in a full version of this paper.

## 3 Finding Subgraphs

In this section, we develop adaptive algorithms for finding four-vertex subgraphs. There are eleven non-isomorphic graphs with 4 vertices: the clique on four vertices ($K_4$) and its complement ($\overline{K_4}$), the diamond ($K_4 - e$) and its complement ($\overline{K_4 - e}$), the claw ($K_{1,3}$) and its complement ($\overline{K_{1,3}}$), the paw ($K_{1,3} + e$) and its complement ($\overline{K_{1,3} + e}$), the cycle on four vertices ($C_4$) and its complement ($\overline{C_4}$), and the path on four vertices ($P_4$) (which is its own complement). Note that $+e$ and $-e$ indicate the insertion of an edge or the deletion of any edge, respectively. A linear-time algorithm is known for detecting whether the input graph contains an induced $P_4$ [12]. For $K_4$ and $\overline{K_4}$, the currently fastest algorithm runs in $O(n^{3.257})$ [18, 21] and for all other graphs the best known algorithm is by Williams et al. [38] and runs in $O(n^\omega)$ time. Their approach can be summarized as follows.

- Let $G$ be an undirected graph and let $A$ be its adjacency matrix. Let $H = (V', E')$ be a four-vertex graph that is none of $K_4$, $\overline{K_4}$, $C_4$, $\overline{C_4}$. There is an integer $2 \leq q_H \leq 6$ such that if we can compute $A^2[u, v]$ for every edge $uv \in E(G)$ in time $t$, then we can compute the number of induced copies of $H$ in $G$ modulo $q_H$ in $O(n + m + t)$ time. See [38, Lemma 4.1] for details. (Some equations provided in [38] require $A^3[v]$ for every $v \in V$. However, this can be computed in $O(m)$ time if $A^2[u, v]$ is given for every edge $uv$.)
- Let $q \geq 2$ be an integer and let $G, H$ be two undirected graphs. Let $G'$ be an induced subgraph of $G$ obtained by independently deleting each vertex with probability $1/2$. If $G$ contains $H$ as an induced subgraph, then the number of induced copies of $H$ in $G'$ modulo $q$ is not 0 with probability at least $2^{-|V(H)|}$. (For our applications, $|V(H)| = 4$, so this probability is at least $1/16$.)

We show that when a $k$-separator is given, one can test in $O(k^{\omega-1}n)$ time whether there is an induced copy of $H$ for each four-vertex graph $H$ except for $K_4$ and $\overline{K_4}$. We start with all graphs except for $K_4$, $\overline{K_4}$, $C_4$, $\overline{C_4}$. Using the framework by Williams et al. [38], it suffices to show how to compute $A^2[u, v]$ for every edge $uv \in E(G)$. Clearly, it requires $\Omega(n^2)$ time to compute the square $A^2$. Our key observation is that the relevant part of $A^2$ can be computed in $O(k^{\omega-1} \cdot n)$ time. (Incidentally, $A^2$ can be computed in $O(k^{\omega-2}n^2)$ time; see Lemma 18.)

▶ **Lemma 4.** *Given a graph $G$ and a $k$-separator $S$, we can compute $A^2[u, v]$ for every edge $uv \in E(G)$ in $O(k^{\omega-1}n)$ time.*

**Proof.** We use the decomposition $(S; T_1, \ldots, T_\nu)$ described in Section 2 and suppose that the adjacency matrix $A$ has the form given in Equation (1). Note that

$$A^2 = \begin{array}{c} \\ S \\ T_1 \\ \vdots \\ T_\nu \end{array} \begin{array}{c} \begin{array}{cccc} S & T_1 & \cdots & T_\nu \end{array} \\ \begin{bmatrix} \zeta & \eta_1 & \cdots & \eta_\nu \\ \eta_1^T & \delta_1 & \cdots & - \\ \vdots & \vdots & \ddots & \vdots \\ \eta_\nu^T & - & \cdots & \delta_\nu \end{bmatrix} \end{array}, \text{ where } \begin{array}{l} \zeta = \gamma^2 + \beta\beta^T, \\ \eta_i = \gamma\beta_i + \beta_i\alpha_i, \text{ and} \\ \delta_i = \beta_i^T\beta_i + \alpha_i^2. \end{array}$$

Note that computing $\zeta$ takes $O(\nu \cdot k^\omega) = O(k^{\omega-1}n)$ time and computing each of the $O(\nu)$ submatrices $\eta_i$ or $\delta_i$ takes $O(k^\omega)$ time. The $-$ represents pairs where the corresponding vertices belong to different $T_i$ and are therefore non-adjacent. We thus do not need to compute these values. Thus, we can compute all relevant values in $O(k^{\omega-1}n)$ time. ◀

By Lemma 4, an induced copy of $H \notin \{K_4, \overline{K_4}, C_4, \overline{C_4}\}$ can be detected in $O(\iota^{\omega-1}n)$ time. We show next that $C_4$ and $\overline{C_4}$ can also be detected in $O(\iota^{\omega-1}n)$ time.

▶ **Proposition 5** (⋆). *Given a graph $G$, a $k$-separator, and a graph $H \in \{C_4, \overline{C_4}\}$, we can test whether $G$ contains $H$ as an induced subgraph in $O(k^{\omega-1}n)$ time.*

Thus, we obtain the following.

▶ **Proposition 6.** *Given a graph $G$, a $k$-separator, and a graph $H$ with four vertices that is not $K_4$ or $\overline{K_4}$, we can test whether $G$ contains an induced copy of $H$ in $O(k^{\omega-1}n)$ time.*

We can also find an induced copy with a constant overhead using a standard self-reduction.

▶ **Corollary 7** (⋆). *Given a graph, a $k$-separator, and a graph $H$ with four vertices that is not $K_4$ or $\overline{K_4}$, we can find an induced copy of $H$ in $O(k^{\omega-1}n)$ time if it exists.*

Finally, let us also remark on the detection of cliques $K_\ell$ and independent sets $\overline{K_\ell}$. Let $t_\ell(n)$ be the time complexity of finding $K_\ell$ (or $\overline{K_\ell}$) on an $n$-vertex graph. (It is known, for instance, that $t_4(n) \in O(n^{3.257})$ [18] using fast rectangular matrix multiplication [21].) Since any $K_\ell$ must fully be contained in $G[S \cup T_i]$ for some $i$, it can be detected in $O(t_\ell(\iota)/\iota \cdot n)$ time. For the detection of $\overline{K_\ell}$, note that if $n/\iota \geq \ell$, then there is an independent set of size $\ell$. Thus, we may assume that $n \leq \iota\ell$. For constant $\ell$, we can thus find $\overline{K_\ell}$ in $O(t_\ell(\iota))$ time.

In the full version of the paper, we also study the problem of finding short(est) cycles. In particular, we show that *girth*, i.e., the length of a shortest cycle, can be computed in $O(\iota^{\omega-1}n)$ time.

## 4 Matching

As mentioned in the introduction, Lovász [30] showed that the cardinality of a maximum matching can be determined in randomized $O(n^\omega)$ time. Several decades later, two algorithms have been developed to find a maximum matching [24, 31]. In this section, we show the following.

▶ **Theorem 8.** *Given a graph and a $k$-separator, we can find a maximum matching in randomized $O(k^{\omega-1}n)$ time.*

In the full version of the paper, we show that the existence of a perfect matching can be checked in $O(k^{\omega-1}n)$ time, where $k$ is the feedback vertex number of the input graph, that is, the minimum number of vertices to delete in order to turn the graph into a forest.

Tutte [36] showed that $G$ has a perfect matching if and only if its symbolic Tutte matrix $A$ is nonsingular. Lovász [30] later showed that the rank of $A$ equals twice the size of maximum matching. To avoid computation over a multivariate polynomial ring, we will assume that each variable $x_{uv}$ is instantiated with an element chosen from a finite field $\mathbb{F}$ uniformly at random. (We will assume that $|\mathbb{F}| = \mathrm{GF}(2^{c\lceil \log n \rceil})$ for a sufficiently large constant $c > 0$.) Since the determinant of the symbolic Tutte matrix has degree at most $n$, by the Schwartz–Zippel lemma [33, 39] (which states that a non-zero polynomial of total degree $d$ over a finite field $\mathbb{F}$ is, when evaluated at a uniformly randomly coordinate, non-zero with probability at least $1 - d/|\mathbb{F}|$), if $G$ has a perfect matching and $\mathbb{F}$ is of size at least $\delta n$, then $A$ is nonsingular with probability at least $1 - 1/\delta$.

Let $A$ be a block matrix $A = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$. If $\alpha$ is nonsingular, then the determinant of $A$ is $\det(A) = \det(\alpha) \cdot \det(C)$, where $C = \delta - \gamma\alpha^{-1}\beta$ is the Schur complement (see e.g., [32]). Thus, assuming that $\alpha$ is nonsingular, $A$ is nonsingular if and only if $\delta - \gamma\alpha^{-1}\beta$ is nonsingular. A simple application of the Schur complement yields the following.

▶ **Lemma 9.** *Consider a matrix $A$ of the following form. Then, provided that $\alpha$ is nonsingular, $A$ is nonsingular if and only if the following matrix $A'$ is nonsingular.*

$$A = \begin{bmatrix} \alpha & \beta & 0 \\ -\beta^T & \gamma & \zeta \\ 0 & -\zeta^T & \eta \end{bmatrix} \qquad A' = \begin{bmatrix} \gamma & \zeta \\ -\zeta^T & \eta \end{bmatrix} - \begin{bmatrix} -\beta^T \\ 0 \end{bmatrix} \alpha^{-1} \begin{bmatrix} \beta & 0 \end{bmatrix} = \begin{bmatrix} \gamma + \beta^T\alpha^{-1}\beta & \zeta \\ -\zeta^T & \eta \end{bmatrix}$$

**Harvey's algorithm.** There is a randomized $O(n^\omega)$-time algorithm to find a perfect matching in a graph (if one exists) due to Harvey [24]. We describe the algorithm outline here. The main idea is to delete edges as long as at least one perfect matching remains until we are left with a graph in which every vertex has exactly one neighbor. Here, deleting an edge

corresponds to setting the corresponding entries in the Tutte matrix to zero. Whether an edge is *deletable*, i.e., whether there remains a perfect matching after its deletion, can be determined in $O(1)$ time after computing the inverse of the Tutte matrix in a preprocessing step. This is essentially due to the fact that for a nonsingular matrix $A$ and column vectors $u$ and $v$, matrix $A + uv^T$ is nonsingular if and only if $1 + v^T A^{-1} u \neq 0$. When an edge is deleted, the inverse needs to be updated and the Sherman-Morrison-Woodbury formula (which is not relevant for us) is employed in Harvey's algorithm. However, it takes $O(n^2)$ time when this update is implemented naively. The crux of Harvey's algorithm lies in a recursive scheme that essentially allows to perform the update in $O(1)$ time. The algorithm uses a subroutine called DeleteEdgesCrossing, which takes two disjoint vertex sets $R$ and $S$ as input and iteratively deletes all deletable edges $uv$ with $u \in R$ and $v \in S$. This subroutine runs in $O(n^\omega)$ time and our algorithm will also use it.

## 4.1 Detecting a perfect matching

In this section, we will describe a randomized $O(k^{\omega-1}n)$-time algorithm to test whether a given graph with a $k$-separator $S$ contains a perfect matching. We assume that the Tutte matrix $A$ has the form of Equation (1) and that it is instantiated randomly from a field $\mathbb{F} = \mathrm{GF}(2^{c\lceil \log n \rceil})$. For each component $T_i$, we find a basis $T_i' \subseteq T_i$ of $A[T_i]$ in $O(k^\omega)$ time. Note that $A[T_i']$ is nonsingular by Lemma 2. Let $T_i^* = T_i \setminus T_i'$, $S_i = S_{i-1} \cup T_i^*$ for $S_0 = S$, and $S^* = S_\nu$. Note that if $G$ has a perfect matching $M$, then at least $|T_i^*|$ vertices of $T_i$ are matched to $S$. Thus, if $\sum_{i\in[\nu]} |T_i^*| > |S|$, we can conclude that there is no perfect matching. Otherwise, we have $|S^*| \leq 2k$. Now, consider for each component $T_i'$ the matrix

$$B_i = \begin{array}{c} \\ T_i' \\ S^* \\ V \setminus (S^* \cup \bigcup_{j\in[i-1]} T_j') \end{array} \begin{array}{c} \begin{array}{ccc} T_i' & S^* & V \setminus (S^* \cup \bigcup_{j\in[i-1]} T_j') \end{array} \\ \begin{bmatrix} \alpha_i & \beta_i & 0 \\ -\beta_i^T & \gamma_i & \zeta_i \\ 0 & -\zeta_i^T & \eta_i \end{bmatrix} \end{array},$$

where $\gamma_1 = \gamma$, $\gamma_{i+1} = \gamma_i + \beta_i^T \alpha_i^{-1} \beta_i$, and all entries except for $\gamma_i$ for $i > 1$ are identical to the corresponding entries of $A$. Note that $\alpha_i$ and $\beta_i$ may differ from $\alpha_i$ and $\beta_i$ in Equation (1), but if $T_j' = T_j$ for all $j \in [\nu]$, then $\alpha_i$ and $\beta_i$ here coincide with $\alpha_i$ and $\beta_i$ from Equation (1). Note that the matrix $B_1$ coincides with the Tutte matrix $A$. Hence, we only need to test whether $B_1$ is nonsingular. We do this by iteratively computing $B_{i+1}$ from $B_i$ in $O(k^\omega)$ time. For notational convenience, we will assume that there is an empty component $T_{\nu+1}'$. Since $\alpha_i = A[T_i']$ is nonsingular, by Lemma 9 the matrix $B_i$ is nonsingular if and only if $B_{i+1}$ is nonsingular. The nonsingularity of $B_{\nu+1} = \gamma_{\nu+1}$ can be tested in $O(k^\omega)$ time since it has size $O(k) \times O(k)$. Note that our algorithm takes $O(k^\omega)$ time for each $i \in [\nu + 1]$ and thus $O(\nu k^\omega) = O(k^{\omega-1}n)$ time overall. If $A$ is nonsingular, then our algorithm correctly concludes that there is a perfect matching. By the Schwartz–Zippel lemma, we know that if $G$ admits a perfect matching, then the probability that $A$ is nonsingular is at least $1 - n/|\mathbb{F}| \geq 1/2$. This leads to the following result.

▶ **Proposition 10.** *Given a graph and a $k$-separator, we can test whether it has a perfect matching in randomized $O(k^{\omega-1}n)$ time.*

In the full version of the paper, we show that whether there is a perfect matching can be checked in $O(k^{\omega-1}n)$ time, where $k$ is the feedback vertex number. We use a similar approach based on the Schur complement. The major difference is that computing $(A[V \setminus S])^{-1}$ would

require $\Omega(n^2)$ time ($A$ is the Tutte matrix and $S$ is a feedback vertex set). Instead, we compute $(A[V \setminus S])^{-1}A[S, V \setminus S]$ via dynamic programming in $O(kn)$ time. To that end, we first show that for entries with both coordinates in $S$ in the Tutte matrix, it suffices to use values in $\{0, 1\}$ instead of variables. This allows us to give a simple characterization of the entries of $(A[V \setminus S])^{-1}$ in terms of alternating paths. Our dynamic program uses a recurrence relation based on this.

▶ **Proposition 11** (⋆). *Deciding whether a given graph $G$ contains a perfect matching can be done in randomized $O(k^{\omega-1}n)$ time, where $k$ is the feedback vertex number of $G$.*

## 4.2 Finding a perfect matching

We next give an algorithm to find a perfect matching (if one exists). We will use the same notation as before and we will again assume that $\alpha_i$ is nonsingular for every $i \in [\nu]$. Note that if the submatrix $\gamma_1$ is also nonsingular, then it is easy to find a perfect matching since the corresponding submatrices $\gamma_1 = A[S^*]$ and $\alpha_i = A[T_i']$ are all nonsingular and therefore there exists a perfect matching in $G[S^*]$ and $G[T_i']$ for every $i \in [\nu]$. We can find these using one of the randomized $O(n^\omega)$-time algorithms [24, 31]. This procedure takes $O(k^\omega\nu) = O(k^{\omega-1}n)$ time. However, in general, we cannot assume that $\gamma_1$ is nonsingular.

To deal with the case that $\gamma_1$ is singular, we use Harvey's aforementioned algorithm. Our algorithm first computes $\gamma_i$ for each $i \in [\nu + 1]$. We then proceed inductively in decreasing order from $i = \nu+1$ to $i = 1$. Our algorithm finds a set $\widetilde{S}_i \subseteq \widetilde{S}_{i+1} \subseteq S^*$ (we set $\widetilde{S}_{\nu+1} = S^*$ for notational convenience) and a matching $M_i$ which saturates every vertex in $(\widetilde{S}_{i+1} \setminus \widetilde{S}_i) \cup T_i'$. The matching $M_i$ is the union of two matchings $M_i'$ and $M_i''$, where $M_i'$ is a perfect matching between $\widetilde{S}_{i+1} \setminus \widetilde{S}_i$ and $T_i' \setminus T_i''$ for $T_i'' \subseteq T_i'$ and $M_i''$ is a perfect matching in $G[T_i'']$ (see Lemmas 13 and 14).

We will maintain the invariant that $\gamma_i[\widetilde{S}_i]$ is nonsingular. For $i = \nu + 1$, we have that $\gamma_{\nu+1}[\widetilde{S}_{\nu+1}] = B_{\nu+1}$ is nonsingular. Moreover, if this invariant is maintained, then $\gamma_1[\widetilde{S}_1]$ is nonsingular and thus $G[\widetilde{S}_1]$ contains a perfect matching that can be combined with all previous $M_i$ to obtain a perfect matching for $G$. To find the set $\widetilde{S}_i$ for $i \in [\nu]$, consider the following submatrix $B_i'$ of $B_i$ whose rows and columns are indexed by $\widetilde{S}_{i+1} \cup T_i'$:

$$B_i' = \begin{bmatrix} \alpha_i & \beta_i[T_i', \widetilde{S}_{i+1}] \\ -\beta_i^T[\widetilde{S}_{i+1}, T_i'] & \gamma_i[\widetilde{S}_{i+1}] \end{bmatrix}.$$

Note that for a nonzero entry in $B_i'$, if one of the coordinates is in $T_i'$, then the corresponding edge exists in $G$ but this is not necessarily the case if both coordinates are in $\widetilde{S}_{i+1}$. We first show that $B_i'$ is nonsingular.

▶ **Lemma 12** (⋆). *For each $i \in [\nu]$, the matrix $B_i'$ is nonsingular.*

Since $B_i'$ is nonsingular, we can compute $(B_i')^{-1}$ and apply the subroutine DeleteEdges-Crossing of Harvey [24] on the bipartition $(\widetilde{S}_{i+1}, T_i')$ in $O(k^\omega)$ time. (This subroutine requires the inverse to be given.) Essentially, we "delete" (i.e., change to zero) all deletable entries with one coordinate in $\widetilde{S}_{i+1}$ and the other in $T_i'$. We then get a skew-symmetric matrix $C_i$ that is identical to $B_i'$ except that some entries in $C_i[\widetilde{S}_{i+1}, T_i']$ and $C_i[T_i', \widetilde{S}_{i+1}]$ are set to zero. Let $\widetilde{S}_i$ be those vertices $v \in \widetilde{S}_{i+1}$ such that $C_i[T_i', v]$ is a zero vector and $T_i''$ be those vertices $u \in T_i'$ such that $C_i[\widetilde{S}_{i+1}, v]$ is a zero vector. Each remaining nonzero entry in $C_i[\widetilde{S}_{i+1}, T_i']$ and $C_i[T_i', \widetilde{S}_{i+1}]$ is undeletable. We show that all edges corresponding to these entries form a perfect matching $M_i'$ between $\widetilde{S}_{i+1} \setminus \widetilde{S}_i$ and $T_i' \setminus T_i''$.

▶ **Lemma 13.** *The set* $M'_i = \{uv \mid C_i[u,v] \neq 0, u \in \widetilde{S}_{i+1} \setminus \widetilde{S}_i, v \in T'_i \setminus T''_i\}$ *is a matching in* $G$ *with high probability.*

**Proof.** For a vertex $u \in \widetilde{S}_{i+1} \setminus \widetilde{S}_i$, assume that there are two vertices $v, v'$ such that $C_i[u,v] \neq 0$ and $C_i[u,v'] \neq 0$. We show that this leads to a contradiction when the variables are treated as indeterminates. Let $\widehat{C_i}_{u,w}$ be the result of deleting the rows and columns indexed by $u$ and $w$ from $C_i$. By Lemma 3, we have

$$\operatorname{pf}(C_i) = C_i[u,v] \operatorname{pf}(\widehat{C_i}_{u,v}) + C_i[u,v'] \operatorname{pf}(\widehat{C_i}_{u,v'}) + p, \tag{2}$$

where $p = \sum_{w \in \widetilde{S}_{i+1} \cup T'_i,\, w \notin \{v,v'\}} C_i[u,w] \operatorname{pf}(\widehat{C_i}_{u,w})$.

By the assumption that deleting the corresponding edge $uv$ or $uv'$ (i.e., setting $x_{uv} = 0$ or $x_{uv'} = 0$) results in a singular matrix (this is due to Harvey's algorithm), we have

$$C_i[u,v] \operatorname{pf}(\widehat{C_i}_{u,v}) + p = 0 \text{ and } C_i[u,v'] \operatorname{pf}(\widehat{C_i}_{u,v'}) + p = 0.$$

We thus obtain $C_i[u,v] \operatorname{pf}(\widehat{C_i}_{u,v}) = C_i[u,v'] \operatorname{pf}(\widehat{C_i}_{u,v'}) = -p$. Note that $p \neq 0$, since otherwise $\operatorname{pf}(C_i) = -p = 0$ by Equation (2), which is a contradiction because $C_i$ is nonsingular by Harvey's algorithm. The left hand side $C_i[u,v] \operatorname{pf}(\widehat{C_i}_{u,v})$ is multiplied by a variable $C_i[u,v] = x_{u,v}$ but this variable does not appear on the right hand side $C_i[u,v'] \operatorname{pf}(\widehat{C}_{u,v'})$. Thus, $Q = C_i[u,v] \operatorname{pf}(\widehat{C_i}_{u,v}) - C_i[u,v'] \operatorname{pf}(\widehat{C_i}_{u,v'})$ is a polynomial that is not identically zero, which is a contradiction. We thus have exactly one vertex $v \in T'_i \setminus T''_i$ such that $C_i[u,v] \neq 0$. Since $Q$ has degree at most $n$, this evaluates to non-zero with high probability by the Schwartz–Zippel lemma. An analogous argument shows that for every $v \in T'_i \setminus T''_i$, there is exactly one $u \in \widetilde{S}_{i+1} \setminus \widetilde{S}_i$ such that $C_i[u,v] \neq 0$. Since every nonzero entry in $C_i[\widetilde{S}_{i+1}, T_i]$ corresponds to an edge in $G$, we are done. ◀

We can now show that our main invariant can be maintained with high probability.

▶ **Lemma 14** (⋆). *The matrices* $\gamma_i[\widetilde{S}_i]$ *and* $\alpha_i[T''_i]$ *are nonsingular with high probability.*

Concluding this section, we prove our main result.

▶ **Proposition 15** (⋆). *Given a graph with a perfect matching and a* $k$*-separator, we can find a perfect matching in* $O(k^{\omega-1}n)$ *time.*

In the full version of the paper, we prove Theorem 8, that is, we show how to find a maximum matching in $O(k^{\omega-1}n)$ time, when given a $k$-separator.

## 5 All-Pairs Shortest Paths

We study the well-known All-Pairs Shortest Paths problem (APSP) on unweighted graphs. APSP on unweighted graphs can be solved in $O(n^\omega)$-time using matrix multiplication [34]. In this section, we show an algorithm which is faster than $O(n^\omega)$ when $k$ is sufficiently small, that is, $k \in O(n^{0.541})$.

▶ **Theorem 16.** *Given an (unweighted undirected) graph and its* $k$*-separator, APSP can be solved in* $O(k^{(\omega-1)/2}n^2) \subseteq O(k^{0.687}n^2)$*-time.*

APSP is closely connected to the min-plus-product. In fact, solving APSP is subcubic equivalent to computing the min-plus product of two matrices [19]. The *min-plus product* $A \star B$ of a $p \times q$-matrix $A$ with a $q \times r$-matrices $B$ is the $p \times r$-matrix $C$

with $C[i,j] = \min_{k\in[q]} A[i,k] + B[k,j]$. While it is conjectured that there is no algorithm computing the min-plus product of two $n \times n$-matrices in $O(n^{3-\epsilon})$ time for any $\epsilon > 0$, there is a subcubic-time min-plus matrix multiplication algorithm for bounded-difference matrices, i.e., matrices in which the difference of two adjacent entries in a row is small [10]. Although the distance matrix $D$ (which contain pairwise distances) does not necessarily have bounded differences, for two adjacent vertices $u$ and $v$, the difference between $D[u,w]$ and $D[u,v]$ for any $w \in V$ is at most 1. The "standard" decomposition of $G - S$ into $T_1, \dots, T_\nu$ from Section 2 is not convenient in this case, since two vertices in $T_i$ may have distance $\Theta(n)$, e.g., when $T_i$ is not connected. Our algorithm first modifies the given graph so that we have bounded-difference distance matrices. The main idea is that, for a $k$-separator $S$, we can join connected components from $G - S$ by adding copies of vertices in $S$. We will also ensure that there is a Hamiltonian path through each component (this step is essentially also used by Deng et al. [14]). This ensures that if we rearrange the rows and columns according to the Hamiltonian path, then two consecutive rows correspond to adjacent vertices and the distances between either of them and some third vertex differ by at most one.

▶ **Lemma 17** ($\star$). *Given a graph $G$ and a $k$-separator $S$, we can compute in $O(n+m)$ time a graph $G'$ and its $O(k)$-separator $S'$ such that $G' - S'$ has connected components $T'_1, \dots, T'_\nu$ for $\nu' \in O(n/k)$, each of which has a Hamiltonian path $H_i$, and $|S'| = |T'_1| = \cdots = |T'_\nu|$. Moreover, given the distance matrix $D'$ for $G'$, one can compute the distance matrix $D$ of $G$ in $O(n^2)$ time.*

Using Lemma 17, we now show that APSP on unweighted graphs can be solved faster than $O(n^\omega)$ if the vertex integrity is sufficiently small and $\omega > 2$.

**Proof of Theorem 16.** If $k \geq n^{0.6}$, then $k^{(\omega-1)/2}n^2 > n^\omega$ and we apply the $O(n^\omega \log n)$-time algorithm [34]. Otherwise, $k \leq n^{0.6}$ and we use the following algorithm to solve APSP.

1. We apply Lemma 17, resulting in a graph $G$ and sets $S, T_1, \dots, T_\nu$.
2. For every $i$, we solve APSP on $G[S \cup T_i]$. Let $D_i = D'_i[T_i, S]$, where $D'_i$ is the distance matrix of $G[S \cup T_i]$.
3. We compute an edge-weighted graph $G_S$ where the vertex set is $S$, the edge set is $\binom{S}{2}$, and $w(uv) = 1$ if $uv \in E(G)$, and $w(uv) = \min_i D_i(uv)$ otherwise. We solve APSP on this weighted graph and call the resulting distance matrix $D_S$. As we show later, $D_S[u,v]$ is the distance from $u$ to $v$ in $G$.
4. For each $i, j \in [\nu]$, we compute $D^*_i := D_i \star D_S$. and $D^*_{i,j} := D^*_i \star D_j^T$.
5. Return a symmetric matrix $D^*$ where the upper triangular part of $D^*$ is defined by

$$D^*[u,v] = \begin{cases} D_S[u,v] & u,v \in S, \\ D^*_i[u,v] & u \in S \text{ and } v \in T_i, \\ \min\{D_i[u,v], D^*_{i,i}[u,v]\} & u,v \in T_i \text{ for some } i \in [\nu], \\ D^*_{i,j}[u,v] & u \in T_i \text{ and } v \in T_j \text{ for } i \neq j. \end{cases}$$

First we show the correctness of the algorithm. For $u, v \in S$, Step 3 guarantees that $D^*[u,v] = D_S[u,v]$ is the distance between $u$ and $v$ in the weighted graph $G_S$. As each edge $u'v'$ in $G_S$ corresponds to a path in $S \cup T_i$ of length $w(u'v')$ for some $i \in [\nu]$, it follows that each path in $G_S$ corresponds to a walk in $G$ of the same length. Thus, $D_S[u,v]$ is not smaller than a shortest $u$-$v$-path in $G$. On the other hand, let $P$ be a shortest $u$-$v$-path in $G$. Let $u = s_1, \dots, s_\ell = v$ be the vertices of $P$ in $S$ (in the order they appear in $P$). By construction, for each $i < \ell$, there are no vertices in $S$ between $s_i$ and $s_{i+1}$ in $P$. Hence, this subpath of $P$ is fully contained in $G[S \cup T_j]$ for some connected component $T_j$ and we

have $w(s_i, s_{i+1}) \leq \text{dist}_{G[S \cup T_j]}(s_i, s_{i+1})$. Consequently, we have that $P$ corresponds to a path in the weighted graph $G_S$ and $D_S(u, v)$ is not larger than the length of a shortest $u$-$v$-path in $G$. Thus, $D_S[u, v]$ equals the distance between $u$ and $v$ in $G$ and $D^*[u, v]$ has therefore been computed correctly.

Next, assume that $u \in T_i$ for some $i \in [\nu]$ and $v \in S$ (the case $u \in S$ and $v \in T_i$ is analogous). A shortest $u$-$v$-path then consists of a shortest $u$-$s$-path in $G[S \cup T_i]$ and a shortest $s$-$v$-path in $G$ for some $s \in S$ (since each vertex in $T_i$ only has neighbors in $T_i \cup S$). This implies $\text{dist}_G[u, v] = \min_{s \in S} D_i[u, s] + D_S[s, v] = D_i^*[u, v]$.

Next, assume that $u \in T_i$ and $v \in T_i$ for some $i \in [\nu]$. Then a shortest $u$-$v$-path either stays completely in $T_i$ or it decomposes into a shortest $u$-$s$-path in $G[S \cup T_i]$, followed by a shortest $s$-$s'$-path in $G$, and finally followed by a shortest $s'$-$v$-path in $G[S \cup T_i]$ for some $s, s' \in S$. In the first case, the distance between $u$ and $v$ equals to $D_i[u, v]$. In the second case, the distance between $u$ and $v$ equals

$$\min_{s,s' \in S} \text{dist}_{G[S \cup T_i]}(u, s) + \text{dist}_G(s, s') + \text{dist}_{G[S \cup T_i]}(s', v) = D_{i,i}^*[u, v].$$

Thus, $D^*[u, v]$ contains the distance between $u$ and $v$.

Finally, assume that $u \in T_i$ and $v \in T_j$ for some $i \neq j \in [\nu]$. A shortest $u$-$v$-path then decomposes into a shortest $u$-$s$-path in $G[S \cup T_i]$, followed by a shortest $s$-$s'$-path in $G$, and finally followed by a shortest $s'$-$v$-path in $G[S \cup T_j]$ for some $s, s' \in S$. Consequently, we have $\text{dist}(u, v) = \min_{s,s' \in S} D_i[u, s] + D_S[s, s'] + D_j[s', v] = D_{i,j}^*[u, v]$.

It remains to analyze the running time of the algorithm. Step 1 runs in linear time by Lemma 17. Step 2 solves APSP on $O(n/k)$ many instances, each with $O(k)$ vertices. As APSP on unweighted graphs with $k$ vertices can be solved in $O(k^\omega \log(k))$ time, Step 2 runs in $O(nk^{\omega-1} \log(k))$ time.

Step 3 first computes a weighted graph on $O(k)$ vertices. Each edge can be computed in $O(n/k)$ time. As there are $O(k^2)$ many edges, computing the graph takes $O(n \cdot k)$ time. Solving weighted APSP on this graph then can be done in $O(k^3)$ time [37]. As we assumed $k \leq n^{0.6}$, this step runs in $O(n^{1.8})$ time.

Step 4 computes for each pair $(i, j) \in [n/k]$ the min-plus product of $D_i$, $D_S$, and $D_j$. We will show that we can compute these min-plus products in $O(k^{(3+\omega)/2})$ time (which is faster than the state-of-the-art algorithm for computing arbitrary min-plus products). The trick is to use the fact that $D_i$ and $D_i^*$ have bounded difference and then use a result by Chi et al. [10] stating that the min-plus product of two matrices of dimension $n \times n$ can be computed in $O(n^{(3+\omega)/2}) \subseteq O(n^{2.687})$ time if one of the matrices has bounded difference. While it is not true a priori that $D_i$ and $D_i^*$ have bounded difference, we will order the rows according to the Hamiltonian path $H_i$. This ensures that two consecutive vertices are adjacent, which implies that consecutive entries in a row differ by at most one. Hence, each computation of the min-plus product can be done in $O(k^{(3+\omega)/2})$ time, as $|S| = |T_1| = \cdots = |T_\nu| = O(k)$ by Lemma 17. Overall, Step 4 runs in $O((n/k)^2 \cdot k^{(3+\omega)/2}) = O(n^2 \cdot k^{(\omega-1)/2})$ time.

Lastly, Step 5 runs in $O(n^2)$ time. The overall running time of $O(k^{(\omega-1)/2} \cdot n^2)$ follows from the observation that the running time for Step 4 dominates the running time of Steps 1, 2, 3, and 5 as $k \leq n$ and $\omega < 2.9$. ◀

**Adaptive algorithm for bounded diameter.** We give an adaptive algorithm for constant diameter. The crucial observation here is that we can compute the product of the adjacency matrix of a graph with any other matrix in $O(\iota^{\omega-2} n^2)$ time.

▶ **Lemma 18.** *Given a graph $G$ with adjacency matrix $A$, a $k$-separator $S$ for $G$, and an $n \times n$-matrix $M$, the matrices $AM$ and $MA$ can be computed in $O(k^{\omega-2} n^2)$ time.*

**Proof.** Suppose that the adjacency matrix has the form as described in Equation (1). Then,

$$AM = \begin{bmatrix} \gamma M[S,V] + \beta M[\overline{S},V] \\ \beta^T M[S,V] + \alpha M[\overline{S},V] \end{bmatrix}.$$

A closer inspection reveals that all the computation can be done in $O(k^{\omega-2}n^2)$ time: First observe that $\gamma M[S,V]$ can be computed in $O(k^{\omega-1}n)$ time. For the other terms, note that

$$\begin{aligned} \beta M[\overline{S},V] &= \begin{bmatrix} \beta M[\overline{S},S] & \beta M[\overline{S},T_1] & \dots & \beta M[\overline{S},T_\nu] \end{bmatrix}, \\ \beta^T M[S,V] &= \begin{bmatrix} \beta^T M[\overline{S},S] & \beta^T M[\overline{S},T_1] & \dots & \beta^T M[\overline{S},T_\nu] \end{bmatrix}, \\ \alpha M[\overline{S},V] &= \begin{bmatrix} \alpha_1 M[T_1,\overline{S}] & \alpha_2 M[T_2,\overline{S}] & \dots & \alpha_\nu M[T_\nu,\overline{S}] \end{bmatrix}^T. \end{aligned}$$

Since there are $O(n/k)$ submatrices and each takes $O(k^{\omega-1}n)$ time to compute, $AM$ can be computed in $O(k^{\omega-2}n^2)$ time. Note that $MA = (AM^T)^T$ can be computed analogously. ◀

We obtain our algorithm from the folklore observation that for any two vertices $u, v \in V$, there is a walk of length exactly length $d$ between $u$ and $v$ if and only if $A^d[u,v] \neq \emptyset$.

▶ **Proposition 19.** *Given a graph $G$ and a $k$-separator $S$, APSP can be solved in $O(dk^{\omega-2}n^2)$ time where $d$ is the diameter of $G$.*

**Proof.** Let $A$ be the adjacency matrix of $G$. We compute matrices $B^1, \dots, B^d \in \{0,1\}^{n \times n}$ recursively as follows. We start with $B^1 := A$. Matrix $B^{i+1}$ is computed by multiplying $B^i$ with $A$ and replacing all non-zero entries by 1. Note that $A^i[u,v] = 0$ if and only if $B^i[u,v] = 0$. Thus, the distance between two vertices $u \neq v$ is the minimum $i$ such that $B^i[u,v] \neq 0$. By Lemma 18, we can multiply any matrix with $A$ in $O(k^{\omega-2}n^2)$ time. Thus, we can compute $B^1, \dots, B^d$ in $O(dk^{\omega-2}n^2)$ time. ◀

Note that $d$ can be of order $\Omega(k^2)$ as the vertex integrity in a cycle with $n$ vertices is in $O(\sqrt{n})$ and its diameter is $\lfloor n/2 \rfloor$.

## 6 Conclusion

In this work, we investigated the parameter vertex integrity in search for more efficient algorithms in the FPT-in-P paradigm. We exhibited that for many problems, the structure of graphs with a small vertex integrity allows us to harness the power of fast matrix multiplication. In particular, we designed randomized $O(\iota^{\omega-1}n)$-time algorithms for finding a four-vertex subgraph (which is not a $K_4$ or a $\overline{K_4}$) and a maximum matching. We also showed that unweighted APSP can be solved in $O(\iota^{(\omega-1)/2}n^2)$ time using min-plus product of bounded-differences matrices, leaving open whether there is an $O(\iota^{\omega-2}n^2)$-time algorithm. More broadly, we wonder whether a similar approach using fast matrix multiplication can be used for graphs of bounded tree-depth or tree-width. Existing approaches (e.g. [11, 20, 26]) do not seem amenable to fast matrix multiplication.

## References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 Noga Alon and Raphael Yuster. Fast algorithms for maximum subset matching and all-pairs shortest paths in graphs with a (not so) small vertex cover. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA '07)*, pages 175–186, 2007. doi:10.1007/978-3-540-75520-3_17.

3 Curtis A Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs - A comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1(38):13–22, 1987.

4 D. Benko, C. Ernst, and Dominic Lanphier. Asymptotic bounds on the integrity of graphs and separator theorems for graphs. *SIAM Journal on Discrete Mathematics*, 23(1):265–277, 2009. doi:10.1137/070692698.

5 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.

6 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph isomorphism on graph classes that exclude a substructure. *Algorithmica*, 82(12):3566–3587, 2020. doi:10.1007/s00453-020-00737-z.

7 Karl Bringmann, Thore Husfeldt, and Måns Magnusson. Multivariate analysis of orthogonal range searching and graph distances. *Algorithmica*, 82(8):2292–2315, 2020. doi:10.1007/s00453-020-00680-z.

8 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974. doi:10.1090/S0025-5718-1974-0331751-8.

9 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*. Springer, 1997. doi:0.1007/978-3-662-03338-8.

10 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22)*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

11 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.

12 Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985. doi:10.1137/0214065.

13 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms*, 15(3):33:1–33:57, 2019. doi:10.1145/3310228.

14 Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. New additive approximations for shortest paths and cycles. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP '22)*, pages 50:1–50:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.50.

15 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.

16 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: Programs with few global variables and constraints. *Artificial Intelligence*, 300:103561, 2021. doi:10.1016/j.artint.2021.103561.

17 Pavlos Eirinakis, Matthew D. Williamson, and K. Subramani. On the Shoshan-Zwick algorithm for the all-pairs shortest path problem. *Journal of Graph Algorithms and Applications*, 21(2):177–181, 2017. doi:10.7155/jgaa.00410.

18 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

**19** Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT '71)*, pages 129–131. IEEE Computer Society, 1971. doi:10.1109/SWAT.1971.4.

**20** Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

**21** François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '12)*, pages 514–523. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.80.

**22** Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.

**23** Chris D. Godsil. *Algebraic combinatorics*. Chapman and Hall, 1993.

**24** Nicholas J. A. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM Journal on Computing*, 39(2):679–702, 2009. doi:10.1137/070684008.

**25** Falko Hegerfeld and Stefan Kratsch. On adaptive algorithms for maximum matching. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 71:1–71:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.71.

**26** Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS '18)*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

**27** Stefan Kratsch and Florian Nelles. Efficient parameterized algorithms for computing all-pairs shortest paths. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS '20)*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.38.

**28** Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC '21)*, pages 34:1–34:15, 2021. doi:10.4230/LIPIcs.ISAAC.2021.34.

**29** Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Mathematical Programming*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.

**30** László Lovász. On determinants, matchings, and random algorithms. In *Proceedings of the 2nd International Symposium on Fundamentals of Computation Theory (FCT '79)*, pages 565–574. Akademie-Verlag, Berlin, 1979.

**31** Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS '04)*, pages 248–255, 2004. doi:10.1109/FOCS.2004.40.

**32** Kazuo Murota. *Matrices and matroids for systems analysis*. Springer Science & Business Media, 1999. doi:10.1007/978-3-642-03994-2.

**33** Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

**34** Raimund Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*, pages 745–749. ACM, 1992. doi:10.1145/129712.129784.

**35** Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*, pages 605–615, 1999. doi:10.1109/SFFCS.1999.814635.

**36** William T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947. doi:10.1112/jlms/s1-22.2.107.

**37** Richard Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

**38** Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '15)*, pages 1671–1680, 2015. doi:10.1137/1.9781611973730.111.

**39** Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the 2nd International Symposium on Symbolic and Algebraic Manipulation (EUROSM '79)*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.

# Approximating Min-Diameter: Standard and Bichromatic

**Aaron Berger**
Massachusetts Institute of Technology, Cambridge, MA, USA

**Jenny Kaufmann**
Harvard University, Cambridge, MA, USA

**Virginia Vassilevska Williams**
Massachusetts Institute of Technology, Cambridge, MA, USA

## Abstract

The *min-diameter* of a directed graph $G$ is a measure of the largest distance between nodes. It is equal to the maximum min-distance $d_{\min}(u, v)$ across all pairs $u, v \in V(G)$, where $d_{\min}(u, v) = \min(d(u, v), d(v, u))$. Min-diameter approximation in directed graphs has attracted attention recently as an offshoot of the classical and well-studied diameter approximation problem.

Our work provides a $\frac{3}{2}$-approximation algorithm for min-diameter in DAGs running in time $O(m^{1.426}n^{0.288})$, and a faster almost-$\frac{3}{2}$-approximation variant which runs in time $O(m^{0.713}n)$. (An almost-$\alpha$-approximation algorithm determines the min-diameter to within a multiplicative factor of $\alpha$ plus constant additive error.) This is the first known algorithm to solve $\frac{3}{2}$-approximation for min-diameter in sparse DAGs in *truly subquadratic* time $O(m^{2-\epsilon})$ for $\epsilon > 0$; previously only a 2-approximation was known. By a conditional lower bound result of [Abboud et al, SODA 2016], a better than $\frac{3}{2}$-approximation can't be achieved in truly subquadratic time under the Strong Exponential Time Hypothesis (SETH), so our result is conditionally tight. We additionally obtain a new conditional lower bound for min-diameter approximation in general directed graphs, showing that under SETH, one cannot achieve an approximation factor below 2 in truly subquadratic time.

Our work also presents the first study of approximating bichromatic min-diameter, which is the maximum min-distance between oppositely colored vertices in a 2-colored graph. We show that SETH implies that in DAGs, a better than 2 approximation cannot be achieved in truly subquadratic time, and that in general graphs, an approximation within a factor below $\frac{5}{2}$ is similarly out of reach. We then obtain an $O(m)$-time algorithm which determines if bichromatic min-diameter is finite, and an almost-2-approximation algorithm for bichromatic min-diameter with runtime $\tilde{O}(\min(m^{4/3}n^{1/3}, m^{1/2}n^{3/2}))$.

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

**Keywords and phrases** diameter, min distances, fine-grained, approximation algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.17

**Related Version** *Full Version*: `https://arxiv.org/abs/2308.08674` [7]

**Funding** *Jenny Kaufmann*: Supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE 2140743.
*Virginia Vassilevska Williams*: Supported by NSF Grant CCF-2129139 and a Sloan Research Fellowship.

## 1 Introduction

The *min-distance* between two vertices $x, y$ in a directed graph $G$ is the minimum of the one-way distances $d(x, y)$ and $d(y, x)$, and is written $d_{\min}(x, y)$. This notion of distance was introduced by Abboud, Vassilevska W., and Wang [2] in their study of diameter in directed graphs. Since the standard notion of distance in directed graphs is not symmetric,




[2] considered a number of symmetric distance functions: roundtrip distance, max-distance, and min-distance. The min-distance in particular, has since then been studied in a series of papers [13, 15, 11].

The min-distance is arguably the most natural notion of distance in directed acyclic graphs (DAGs), in which for every two vertices $x, y$, at most one of $d(x, y)$, $d(y, x)$ is finite. Min-distance is also applicable in potential real-world contexts: for example, if a patient needs to see a doctor as soon as possible, the doctor can visit the patient or vice versa.

The *min-diameter* of a directed graph $G$ is the maximum min-distance between any two vertices, or $\max_{x,y \in V(G)} d_{\min}(x, y)$. One can additionally define the *bichromatic min-diameter* in a graph $G$ with vertex set $V = A \sqcup B$ partitioned into "red" and "blue" vertices: the bichromatic min-diameter is the maximum min-distance between oppositely-colored vertices, or $\max_{a \in A, b \in B} d_{\min}(a, b)$. These are variants on the standard notions of diameter [1, 3, 6, 10, 12, 16, 24] and bichromatic diameter [5, 14], respectively.

One can compute min-diameter or bichromatic min-diameter – and for that matter All Pairs Shortest Paths (APSP), the shortest path distances $d(x, y)$ for all $x, y \in V$ – in an $n$-vertex $m$-edge graph in $O(mn + n^2 \log n)$ time, simply by running Dijkstra's algorithm from every vertex. In unweighted graphs, one can instead use BFS, giving an $O(mn)$ runtime.

One might ask whether, for either min-diameter or bichromatic min-diameter, a faster algorithm exists. However, just as for standard diameter and other diameter variants that have been studied [2, 14, 20], the Strong Exponential Time Hypothesis (SETH) [18, 9] suggests exact computation cannot be done in runtimes that are *truly subquadratic*, meaning $O(m^{2-\epsilon})$ for $\epsilon > 0$. SETH is one of the main hypotheses in Fine-Grained Complexity [22], and is among the most well-established hardness hypotheses for showing conditional lower bounds. It states that for every $\epsilon > 0$, there is an integer $k \geq 3$ so that $k$-SAT on $n$ variables cannot be solved in $O(2^{(1-\epsilon)n})$ time.

Since exact computation is conditionally hard, we resort to finding approximations. Since min-distance does not obey the triangle inequality, approximating min-diameter is especially challenging, in comparison to standard diameter or even roundtrip diameter or max-diameter. For this reason, much of the work on min-diameter has focused on DAGs.

## Min-Diameter

Abboud, Vassilevska W., and Wang [2] gave a 2-approximation for min-diameter in DAGs, running in $\tilde{O}(m)$ time.[1] Meanwhile, they showed that if one can obtain a $(\frac{3}{2}-\delta)$ approximation in $O(m^{2-\epsilon})$ time for min-diameter in DAGs (for $\varepsilon, \delta > 0$), then SETH is false. The results of [2] left a gap between the conditional lower bound of 3/2 and the upper bound of 2 for $O(m^{2-\epsilon})$ time algorithms for DAGs. (The lower bound is for $O((mn)^{1-\epsilon})$ time algorithms but since the instances are sparse, this is the same as $O(m^{2-\epsilon})$ time.)

Later work by Dalirrooyfard and Kaufmann [13] showed that in *dense* DAGs, one can beat the $mn$ barrier by obtaining an almost-$\frac{3}{2}$-approximation algorithm running in $O(n^{2.35})$ time. The conditional lower bound of [2] is for sparse DAGs, however, and the gap between upper and lower bounds has remained.

Our main result is to close this gap for sparse DAGs:

▶ **Theorem 1.** *There is an $O(m^{0.713}n)$ time algorithm that achieves a $(\frac{3}{2}, \frac{1}{2})$-approximation for min-diameter in any $m$-edge, $n$-node unweighted DAG.*

*Furthermore, there is an $O(m^{1.426}n^{0.288})$ time algorithm that achieves a $\frac{3}{2}$-approximation for min-diameter in any $m$-edge, $n$-node unweighted DAG.*

[1] The tilde hides polylogarithmic factors.

A $(c, a)$-approximation for a quantity $D$ is a quantity $D'$ such that $D \leq D' \leq cD + a$.

Similar to [13], we use hitting set and set intersection methods to certify distances. Our main new technique is to iteratively grow a central interval of vertices with convenient distance properties by checking that at least one of two neighboring vertex subsets, to its left and right, has the desired properties.

The algorithms use fast matrix multiplication. There are also combinatorial versions of the algorithms, with runtimes $O(m^{3/4}n)$ for the $(\frac{3}{2}, \frac{1}{2})$-approximation and $O(m^{5/4}n^{1/2})$ for the $\frac{3}{2}$-approximation.[2] These runtimess are still subquadratic for sparse graphs.

Our paper also considers the case of general directed graphs. Abboud, Vassilevska W., and Wang [2] showed that under SETH, any $O(m^{2-\varepsilon})$ time algorithm for $\varepsilon > 0$ for min-diameter can achieve at best a 2-approximation. This result only held for weighted graphs.

We give the first hardness result for unweighted graphs, extending the hardness of [2]:

▶ **Theorem 2.** *Under SETH, there can be no $O(m^{2-\varepsilon})$ time $(2-\delta)$-approximation algorithm for the min-diameter of an unweighted directed graph with $n$ vertices and $m = n^{1+o(1)}$ edges, for $\epsilon, \delta > 0$.*

Because our min-diameter approximation algorithms for DAGs obtain an approximation factor better than 2 in truly subquadratic time, this gives the first separation of hardness results for min-diameter approximation in the sparse cyclic versus acyclic cases.

This result, along with all other hardness results from SETH in this work, are via Orthogonal Vectors (OV) reductions. The OV problem is as follows: Given two sets $A, B$ each containing $n$ $d$-dimensional Boolean vectors, determine whether there are vectors $a \in A$ and $b \in B$ that are *orthogonal*, meaning $a \cdot b = 0$.

▶ **OV Conjecture** ([23]). *There is no constant $\epsilon > 0$ such that for any constant $c$ and $d = c \log n$, the OV problem can be solved by a randomized algorithm in time $O(n^{2-\epsilon})$.*

The OV Conjecture is implied by SETH [23]. OV-based sparse graph constructions have been commonly used to provide hardness results for diameter variants for the past decade, after having been first introduced by Roditty and Vassilevska W. in [20]. However, unlike previous OV-based constructions known to us, our construction is of a graph shaped as a cycle. If there are no orthogonal vectors then all vertices can reach one another via a single loop around the cycle, whereas if there are orthogonal vectors a path between them must loop nearly twice around the cycle, giving a min-diameter nearly twice as large.

As for upper bounds for general directed graphs, Dalirrooyfard et al. [15] gave for every integer $k \geq 2$, an $\tilde{O}(mn^{1/k})$ time $(4k-5)$-approximation algorithm for min-diameter. Most recently, Chechik and Zhang [11] achieved a 4-approximation in near-linear time. There is still a gap between the best approximation known in $O(m^{2-\varepsilon})$ time (3 by [15]) and the best hardness result for such algorithms (2 by this work for unweighted, and by [2] for weighted).

## Bichromatic Min-Diameter

A *bichromatic* version of diameter was considered by Dalirrooyfard et al. [14]. In a graph whose nodes are colored red and blue, the *bichromatic diameter* is the largest distance between two nodes of different colors. Dalirrooyfard et al. [14] gave algorithms and hardness for bichromatic diameter under the usual notion of distance.

[2] Combinatorial is here used to refer to an algorithm that does not rely on fast matrix multiplication.

Our work presents the first study of the notion of bichromatic min-diameter. As the min-diameter is the most natural diameter notion for DAGs, the bichromatic min-diameter is likewise a natural way to consider distances in DAGs whose vertices are two-colored.

We first give hardness for general directed graphs, suggesting that bichromatic min-diameter may be harder than regular min-diameter:

▶ **Theorem 3.** *Under SETH, there can be no $O(m^{2-\varepsilon})$ time $(\frac{5}{2}-\delta)$-approximation algorithm for bichromatic min-diameter in unweighted $n$-node, $m = n^{1+o(1)}$-edge graphs for $\epsilon, \delta > 0$.*

*Furthermore, under SETH, there can be no $O(m^{2-\varepsilon})$ time $(3-\delta)$-approximation algorithm for bichromatic min-diameter in graphs with $O(\log n)$ bit integer edge weights.*

It would be interesting if the $\tilde{O}(m\sqrt{n})$ 3-approximation algorithm for min-diameter of [15] can be extended to work for bichromatic min-diameter, as then one could get a tight result in the weighted case.

We next turn to bichromatic min-diameter in DAGs. We give an almost-2-approximation algorithm and show that it is essentially tight (up to the additive error) under SETH:

▶ **Theorem 4.** *Under SETH, there can be no $O(m^{2-\varepsilon})$ time $(2-\delta)$-approximation algorithm for bichromatic min-diameter in unweighted $n$-node, $m = n^{1+o(1)}$-edge DAGs, for $\epsilon, \delta > 0$.*

▶ **Theorem 5.** *There is an $\tilde{O}(m^{4/3}n^{1/3})$-time algorithm, which, given a DAG and maximum red-blue edge weight $M$, outputs a $(2, M)$-approximation of the bichromatic min-diameter.*

Finally, we present a linear-time algorithm which determines whether a directed graph has finite bichromatic min-diameter. The proof may be found in the full version of the paper [7].

▶ **Theorem 6.** *There is an $O(m)$ time algorithm which checks, for any weighted directed graph $G$, whether the bichromatic min-diameter is finite.*

## 1.1 Preliminaries

We assume the word-RAM model of computation with $O(\log n)$ bit words. All of our algorithms and reductions fall within this model.

Graphs in this work are directed and weakly connected.[3] Edge weights are polynomial in $n$.

The min-eccentricity of a vertex, $\epsilon(v)$, is given by $\max_{u \in V} d_{\min}(u, v)$.

For $v \in V$, the *distance-$D$ out-neighborhood* of $v$ is $N_D^{out}(v) = \{w \in V \setminus \{v\} \mid d(v, w) \leq D\}$. We define $N_D^{in}(v)$ correspondingly.

Given a DAG $G$ with topological ordering $\pi$ and vertex sets $S, T \subseteq V = V(G)$, we write $S <_\pi T$ if all vertices in $S$ appear to the left of all vertices in $T$. When $S$ or $T$ is equal to $\{x\}$ for some vertex $x$, we may omit the brackets. For vertices $s, t$, we write $s \leq_\pi t$ if $s <_\pi t$ or $s = t$. We define a closed subset (with respect to $\pi$) to be a subset $S$ such that for all $v \in V$, either $v \in S$, $v >_\pi S$, or $v <_\pi S$.

Given a DAG $G$ with topological ordering $\pi$, a vertex $v \in V$ and a set $S \subseteq V$, let $s_v \in S$ be the left-most vertex in $S$ such that $d(v, s) \leq D$, if such an $s_v$ exists. Then we define $N_{D,S}^{out}(v)$ to be the set of vertices $w$ such that $d(v, w) \leq D$ and, if $s_v$ exists, $w \leq_\pi s_v$. One can intuitively think of $N_{D,S}^{out}(v)$ as the set $N_D^{out}(v)$ of vertices at distance at most $D$ from $v$,

[3] If a graph is not weakly connected (which can be checked in $O(m + n)$ time), then it has infinite min-diameter, as well as infinite bichromatic min-diameter if its vertices are 2-colored.

but cut off after the first (left-most) time we hit $S$. We define $N^{in}_{D,S}(v)$ symmetrically. A set $S$ such that for all $v$, $|N^{out}_{d,S}(v)|, |N^{in}_{d',S}(v)| \leq k$, will be called a $(k,(d,d'))$-*neighborhood cover*. If $d = d'$ we refer to it as a $(k,d)$-neighborhood cover.

A *bichromatic DAG* $G$ is a DAG whose vertices are two-colored. An $(A,B)$-*separated DAG*, which we may also simply call a *separated DAG* is a DAG ordered according to some topological ordering $\pi$ with color sets $A, B$ such that $A <_\pi B$.

We sometimes omit $\pi$ when the choice of $\pi$ is clear.

Let $\omega(1,r,1)$ be the exponent of the runtime of multiplying $n \times n^r$ by $n^r \times n$ matrices. The square matrix multiplication exponent is $\omega = \omega(1,1,1) > 2.37286$ [4].

## 1.2 Techniques

In this section, we review two useful techniques. The first of these is the greedy set cover lemma. This lemma, and a related randomized version, have been commonly used in prior work on diameter variants ([3], [20], [8], [2], [13]). See [21] for a proof of the lemma.

▶ **Lemma 7.** *Let $p = O(n)$, and let $X_1, \ldots X_p \subseteq V$ have size $|X_i| \geq n^\epsilon$ for $\epsilon \in [0,1]$. In time $O(n^{1+\epsilon})$ one can construct a set $S \subseteq V$ of size $O(n^{1-\epsilon} \log n)$ such that $S \cap X_i \neq \varnothing$ for all $i \in [p]$.*

The following technique, previously used in [13], constructs a set cover of all sufficiently large balls of radius $d$.

▶ **Lemma 8.** *Given a topologically ordered DAG $G$ and parameters $d, d'$ and $k = n^\epsilon$ for $\epsilon \in [0,1]$, one can in time $O(nk^2)$ construct a $(k,(d,d'))$-neighborhood cover $S$ of size $O(\frac{n}{k} \log n)$, and also construct the sets $N^{out}_{d,S}(v)$, $N^{in}_{d',S}(v)$ for all $v \in V$.*

**Proof.** For each vertex $v$, if $|N^{out}_d(v)| < k$, we define $X^k_d(v) = N^{out}_d(v)$. Otherwise let $X^k_d(v)$ be the $k$ left-most vertices in $N^{out}_d(v)$. Similarly, we define the set $Y^k_{d'}(v)$: if $|N^{in}_{d'}(v)| < k$, then $Y^k_{d'}(v) = N^{in}_{d'}(v)$, and otherwise $Y^k_{d'}(v)$ is the $k$ right-most vertices in $N^{in}_{d'}(v)$.

We can compute $X^k_d(v)$ as follows: We initialize a set $S_0 = \varnothing$. While $|S_t| < k$, at step $t$, we consider the set $W = N^{out}(S_t \cup \{v\}) \cap N^{out}_d(v)$ of out-neighbors of $S_t \cup \{v\}$ that are at distance at most $d$ from $v$. If $W$ is nonempty, we let $w$ be its left-most element, and we construct $S_{t+1} = S \cup \{w\}$. If $W$ is empty, then $S_t = N^{out}_d(v) = X^k_d(v)$. If $|S_t| = k$, then $S_t$ contains the left-most $k$ vertices in $N^{out}_d(v)$. Thus, in either case, we will eventually construct $S_t = X^k_d(v)$. The key step in this construction process, namely finding $w$, can be done by maintaining a list containing the left-most neighbor of each vertex in $S_t$ such that the neighbor is of distance at most $d$ from $v$, and choosing the left-most vertex in the list at each step. At step $t$, this list has length at most $|S_t| < k$, and the total number of steps is at most $k$, so the construction can be done in $O(k^2)$ time.

We can construct $Y^k_{d'}(v)$ in $O(k^2)$ time similarly. Doing this for all $v \in V$ takes time $O(nk^2)$.

Lemma 7 gives us a set cover $S$ of size $O(\frac{n}{k} \log n)$ which intersects all sets $X^k_d(v), Y^k_{d'}(v)$ of size at least $k$. This takes time $O(nk)$. Then for each vertex $v$, we construct $N^{out}_{d,S}(v)$ as the set obtained from $X^k_d(v)$ by removing all vertices to the right of the left-most $s \in S \cap X^k_d(v)$. We construct $N^{in}_{d,S}(v)$ in a symmetric fashion. Since the sets $X^k_d(v), Y^k_{d'}(v)$ were of size at most $k$, the sets $N^{out}_{d,S}(v), N^{in}_{d',S}(v)$ are also of size at most $k$. ◀

# 2 Min-diameter approximation

In Section 2.1 we present a conditional lower bound showing that the OV Conjecture implies that no $(2-\delta)$-approximation algorithm for min-diameter in unweighted graphs can run in truly subquadratic time. Subsequently, in Section 2.2, we give an almost-$\frac{3}{2}$-approximation algorithm for min-diameter in unweighted DAGs which runs in truly subquadratic time.

## 2.1 Conditional lower bound in general graphs

We first present a conditional lower bound for approximating min-diameter in general graphs.

▶ **Theorem 9.** *If there are $\epsilon, \delta > 0$ such that there is an $O(m^{2-\epsilon})$-time $(2-\delta)$-approximation algorithm for min-diameter, then the OV Conjecture is false.*

**Proof.** Given $\delta > 0$, choose $t = \lceil \frac{2}{\delta} \rceil$ so that $2 - \delta < \frac{2t}{t+1}$. Let $A$ be an instance of single-set OV[4], that is, a set of $n \geq 2^t$ vectors in $\{0,1\}^{c_0 \log n}$ for a constant $c_0 > 0$. We will construct a graph $G_t$ with $O(tn)$ vertices, $\tilde{O}(tn)$ edges such that the min-diameter is $2t+1$ if $A$ contains a pair of orthogonal vectors, and $t+1$ otherwise. For each $i \in [t]$, we construct a set $A_i$ of $n$ vertices corresponding bijectively to the vectors in $A$. For each vector $a \in A$, let $a_i$ be the vertex in set $A_i$ corresponding to $a$. We also construct a set $I$ of "index" vertices $x_1, \ldots, x_{c \log n}$. In total we have $O(tn)$ vertices. We add an edge $a_1 \to x_j$ and an edge $x_j \to a_t$ whenever $a[j] = 1$. We also add edges $a_i \to a_{i-1}$ for each $i \in \{2, \ldots, t\}$. Finally, we add all possible edges $A_2 \to I$. In total we have $\tilde{O}(tn)$ edges. This graph $G_t$ may be constructed in $\tilde{O}(tn) = \tilde{O}(n)$ time.

We now check that if the OV instance is a YES instance, $G_t$ has at least min-diameter $2t+1$, and in the NO case, $G_t$ has min-diameter $t+1$.

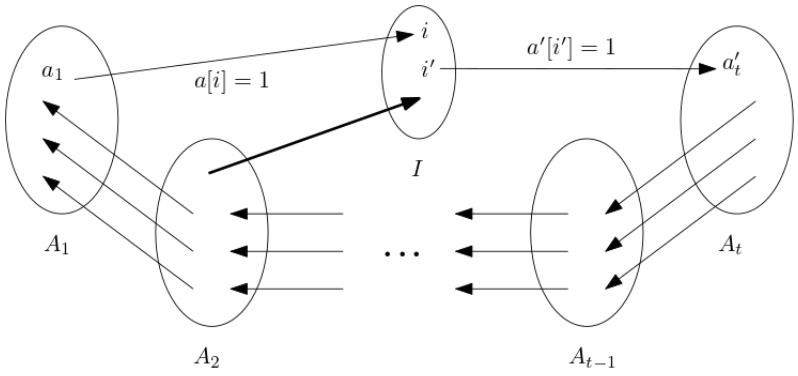



**Figure 1** The graph $G_t$. The thick edge denotes that all possible edges $A_2 \to I$ exist.

### YES case

Let $a, b \in A$ be orthogonal. Without loss of generality, $d_{\min}(a_1, b_1) = d(a_1, b_1)$. There is no $j$ such that $a[j] = b[j] = 1$, so there is no length-2 path from $a_1$ to $b_t$ via $I$. Since all edges between $A_i$ and $A_{i-1}$ are of the form $a_i \to a_{i-1}$, the first $t$ vertices on any $a_1 \to b_1$ path

[4] The OV Conjecture can be equivalently stated in terms of single-set OV (OV where $A = B$). Informally, the reduction is to construct $A'$ from $A$ by appending 10 to all vectors, and $B'$ from $B$ by appending 01 to all vectors. If $v_1, v_2 \in A' \cup B'$ are orthogonal, then we must have $v_1 \in A'$, $v_2 \in B'$ or vice versa.

must be of the form $a_1 \to x_j \to c_t \to \cdots \to c_2$ for some $j \in I, c \in A$ such that $a[j] = c[j] = 1$. We note that any path from $c_2$ to $b_1$ must traverse all sets of the cycle with the possible exception of $A_1$, so must be of length at least $t$. Then if $k$ is an index such that $b[k] = 1$, the path $c_2 \to x_k \to b_t \to \cdots \to b_1$ gives $d(c_2, b_1) = t + 1$. Thus, $d(a_1, b_1) = d_{\min}(a_1, b_1) \geq 2t + 1$. So $G_t$ has min-diameter at least $2t + 1$.

### NO case

Suppose that $A$ contains no pair of orthogonal vectors. For any $a_1, b_i$, there is some $j$ such that $a[j] = b[j] = 1$, so there is a path $a_1 \to x_j \to b_t \to \ldots b_i$ of length at most $t + 1$. Now, consider any vertices $a_i, b_{i'}$ with $2 \leq i \leq i'$. Let $j$ be an index such that $b[j] = 1$. There is a path $a_i \to \ldots a_2 \to x_j \to b_t \to \cdots \to b_{i'}$ which has length at most $t$. Furthermore, for any $a_1 \in A_1$, $x_j \in I$, $d(a_1, x_j) \leq t + 1$, since there is a path $a_1 \to x_k \to a_t \to \cdots \to a_2 \to x_j$ for some $k$ with $a[k] = 1$. Lastly, for any $a_i \in A_i$ with $i \geq 2$, and for any $x_j \in I$, there is a path $a_i \to \ldots a_2 \to x_j$ of length at most $t$. Hence, all pairs of vertices are at min-distance at most $t + 1$. ◀

## 2.2 A $(\frac{3}{2}, \frac{1}{2})$-approximation algorithm in unweighted DAGs

Unlike in general directed graphs, where $(2 - \delta)$-approximating min-diameter seems to be hard, in DAGs one can achieve an efficient $(\frac{3}{2}, \frac{1}{2})$-approximation and a (slightly less) efficient $\frac{3}{2}$-approximation. The $(\frac{3}{2}, \frac{1}{2})$ and the exact $\frac{3}{2}$ approximations have very similar proofs; we present the proof of the $(\frac{3}{2}, \frac{1}{2})$-approximation result here because the algorithm is faster and marginally simpler than in the exact $\frac{3}{2}$ case, whose proof can be found in the full version of the paper [7].

Our algorithm uses fast sparse matrix multiplication to compute set intersections. Its runtime involves the constants $\alpha = \max\{0 \leq r \leq 1 \mid \omega(1, r, 1) = 2\}$ and $\beta = \frac{\omega - 2}{1 - \alpha}$.

▶ **Theorem 10.** *There is an $\tilde{O}(m^{\frac{4\beta+2-2\alpha\beta}{5\beta+3-2\alpha\beta}} n)$-time algorithm achieving a $(\frac{3}{2}, \frac{1}{2})$-approximation for min-diameter in unweighted DAGs.*

Since $\alpha > 0.31389$ [17] and $\omega < 2.37286$ [4], we can use $\beta \simeq 0.5435$, giving the runtime $O(m^{0.713} n)$.

The algorithm starts by topologically sorting the DAG and fixing a neighborhood-size parameter $k$ (whose value will be determined later). It then performs two layers of recursion. The outer layer is a binary search over min-diameter estimates $D$: for each estimate $D$, it will either determine that $D$ is low or high, i.e. that $\texttt{min-diameter}(G) > D$ or $\texttt{min-diameter}(G) \leq \lceil \frac{3}{2} D \rceil$. The inner layer, in which the estimate $D$ is fixed, involves recursively splitting the graph in half according to the topological ordering. Then, in the core body of the algorithm, one of the following will occur:

- We find a pair of vertices with min-distance larger than $D$, in which case we end the recursion and report that $\texttt{min-diameter}(G) > D$.
- We verify that every min-distance between a vertex in the left half and a vertex in the right half is at most $\lceil 3D/2 \rceil$, and we recurse on the left and right halves of the graph.

The core body of the algorithm – which will be repeated recursively as the graph is repeatedly cut in half – is as follows:

First, using Lemma 8, we compute a $(k, (D/2, \lceil D/2 \rceil))$-neighborhood cover $S$ of size $O(\frac{n}{k} \log n)$ along with neighborhoods $N^{out}_{D/2,S}(v), N^{in}_{\lceil D/2 \rceil,S}(v)$ all of which have size at most $k$. Using BFS, we check that $d_{\min}(s, v) \leq D$ for all $s \in S, v \in V$. This step ensures that, for

every vertex with a "sufficiently large" (meaning, size-$k$) distance-$D/2$ out-neighborhood or distance-$\lceil D/2 \rceil$ in-neighborhood, some vertex in $S$ lies in that out- or in-neighborhood. In this sense, $S$ covers all of the large neighborhoods, which will be useful later.

We partition $V$ into $2\theta = n/k^2$ closed intervals, $V_1, \dots V_{2\theta}$. We let $L = V_1 \cup \cdots \cup V_\theta$ and $R = V_{\theta+1} \cup \cdots \cup V_{2\theta}$ be the left and right halves of $V$. We start in the middle and work outwards verifying that min-distances between vertices in $L$ and $R$ are at most $\lceil 3D/2 \rceil$.

At each inductive step, we consider two subsets of vertices $A = V_i$ and $B = V_j$ to the left and right of the "middle interval," which consists of the intervening subsets $V_{i+1} \cup \cdots \cup V_{j-1}$. Intuitively, one might picture the middle interval as a growing amoeba, which at each step engulfs one of $A$ or $B$. We will only add vertices to the middle interval once they have been "checked," so vertices from $A$ that are added to the middle interval have distance at most $3D/2$ to everything in $B$, and vice-versa. Eventually, either the algorithm will detect a min-distance greater than $D$ and report that `min-diameter(`$G$`)` $> D$, or the amoeba will have engulfed enough of the graph that it contains the entirety of one of the halves $L$ or $R$, meaning that all distances from vertices in $L$ to vertices in $R$ are at most $\lceil \frac{3}{2} D \rceil$.

In order to expand the middle interval we have to confirm that one of our two candidate subsets $A$ and $B$ has small distances to the opposite half of the graph. Performing a BFS from each vertex in $A$ or $B$ to confirm this directly would be too slow, so instead we present two subroutines to achieve this goal faster. Algorithm 2 checks that all min-distances between vertices $a \in A$ and $b \in B$ are at most $\lceil 3D/2 \rceil$. Algorithm 3 checks that either all vertices in $A$ have min-distance at most $\frac{3}{2}D$ to all vertices to the right of $B$, or that a symmetric property holds for all vertices in $B$. These algorithms will either detect that the min-diameter is more than $D$ or allow us to add one of $A$ or $B$ to the middle interval.

We first give a pseudocode description of the main algorithm, Algorithm 1, which will give context for the two subroutines, Algorithms 2 and 3, whose descriptions follow afterwards.

We will present the correctness and runtime analyses for the two subroutines, and then present the correctness proof and runtime analysis for the overall algorithm.

The first subroutine, Algorithm 2, checks distances between pairs of vertices in subsets $A, B$ by using the cover $S$ to hit large neighborhoods and using fast sparse rectangular matrix multiplication to efficiently check set intersections between small neighborhoods.

▶ **Theorem 11** ([19]). *If $M, M'$ are $p \times l$ matrices having at most $l$ nonzero entries, where $p^{1+\frac{\alpha}{2}} \le l \le p^{\frac{\omega+1}{2}}$, then in time $O(l^{\frac{2\beta}{\beta+1}} p^{\frac{2-\alpha\beta}{\beta+1}})$ one can compute the product $M^T M'$.*[5]

▶ **Lemma 12.** *Algorithm 2 produces the correct output in runtime $O(k^{4+\frac{2\beta-2\alpha\beta}{\beta+1}})$.*

**Proof.** We first show correctness. Assume the algorithm fails; then it fails inside the **foreach** loop at some pair $a, b$. Suppose for the sake of contradiction that $d_{\min}(a,b) \le D$. Let $x$ be a midpoint of the shortest path from $a$ to $b$, so $d(a,x) \le D/2$, $d(x,b) \le \lceil D/2 \rceil$. If $x \notin N^{out}_{D/2,S}(a)$ despite being distance $\le D/2$ from $a$, then $x$ must lie after the out-neighborhood of $a$ reaches the $(k, (D/2, \lceil D/2 \rceil))$-neighborhood cover $S$ and is cut off. That is, there is some $s \in N^{out}_{D/2}(a) \cap S$ to the left of $x$, and hence to the left of $b$. Thus the condition in line 9 holds, and we **continue** rather than **FAIL**, which is a contradiction. Similarly, if $x \notin N^{in}_{\lceil D/2 \rceil,S}(b)$ then the condition in line 7 holds, which is a contradiction. We conclude that $x \in N^{out}_{D/2,S}(a) \cap N^{in}_{\lceil D/2 \rceil,S}(b)$, in which case $M_{ab} \ge 1$, completing the contradiction. Thus, $d_{\min}(a,b) > D$, and so `min-diameter`$(G) > D$.

[5] To be precise, if we have $\alpha \ge a, \omega \le c$, we can define $b = \frac{c-2}{1-a}$, and then the theorem holds for any such pair $a, b$, used in place of $\alpha, \beta$.

**Algorithm 1** Full graph min-distance tester.

**Input:** DAG $G = (V, E)$, diameter guess $D$, parameters $k = m^{\frac{\beta+1}{5\beta+3-2\alpha\beta}}$ and $\theta = n/2k^2$.

**Output:** One of the following. Each output verifies a corresponding property of $G$.

| | | |
|---|---|---|
| **PASS** | $\Rightarrow$ | $\texttt{min-diameter}(G) \leq \lceil \frac{3}{2}D \rceil$ |
| **FAIL** | $\Rightarrow$ | $\texttt{min-diameter}(G) > D$ |

1 Topologically sort $G$;
2 Using Lemma 8 compute a $(k, (D/2, \lceil D/2 \rceil))$-neighborhood cover $S \subseteq V$. Run BFS to and from each vertex in $S$;
3 **if** $\exists s \in S$ *such that* $\epsilon(s) > D$ **then**
4  **FAIL**
5 Partition $V$ into consecutive closed intervals, $V_1, \ldots V_{2\theta}$, with $|V_i| = k^2$ for each $i$;
6 Initialize $i = \theta$ and $j = \theta + 1$;
7 **while** $i \geq 1$ **and** $j \leq 2\theta$ **do**
8  Run Algorithm 2 (all-pairs) on the pair $(V_i, V_j)$. If this fails, then **FAIL**;
9  Run Algorithm 3 (directional tester) on the pair $(V_i, V_j)$. If this fails, then **FAIL**;
10  **else if** *Algorithm 3 passes and returns* $V_i$ **then**
11  $i = i - 1$;
12  **else**
 /* Otherwise, Algorithm 3 passes and returns $V_j$. */
13  $j = j + 1$;
/* If this line is reached, all distances from $L$ to $R$ are at most $\lceil \frac{3}{2}D \rceil$. */
14 Recursively call this algorithm on $G[L]$ and $G[R]$. If either fails, **FAIL**. Else **PASS**;

Conversely, assume the algorithm passes. Then for each pair of vertices $a \in A, b \in B$, one of the conditions in lines 5, 7, or 9 must hold. If $M_{ab} \geq 1$, then there is some $x \in N^{out}_{D/2,S}(a) \cap N^{in}_{\lceil D/2 \rceil,S}(b)$, giving $d(a,b) \leq d(a,x) + d(x,b) \leq D/2 + \lceil D/2 \rceil \leq D+1$. Otherwise if $a <_\pi N^{in}_{\lceil D/2 \rceil,S}(b)$ and $N^{in}_{\lceil D/2 \rceil,S}(b) \cap S$ is nonempty, then there is some $s \in N^{in}_{\lceil D/2 \rceil,S}(b) \cap S$ with $s \geq_\pi a$. Thus $d(a,b) \leq d(a,s) + d(s,b) \leq D + \lceil D/2 \rceil \leq \lceil 3D/2 \rceil$. Similarly, if $b >_\pi N^{out}_{D/2,S}(a)$ and $N^{out}_{D/2,S}(a) \cap S$ is nonempty, then there is some $s \in N^{out}_{D/2}(a) \cap S$ with $s \leq_\pi b$. So $d(a,b) \leq d(a,s) + d(s,b) \leq D/2 + D \leq 3D/2$. We conclude that if the algorithm passes, then every pair $a \in A, b \in B$ satisfies $d(a,b) \leq \lceil 3D/2 \rceil$.

We conclude with runtime analysis. $M_a$ and $M_b$ are sparse $p \times n$ matrices with $O(pk) = O(k^3)$ entries, so may be treated as $p \times pk$ matrices. Using Theorem 11, the matrix multiplication takes time at most $O((pk)^{\frac{2\beta}{\beta+1}} p^{\frac{2-\alpha\beta}{\beta+1}}) = O(k^{4+\frac{2\beta-2\alpha\beta}{\beta+1}})$. The **foreach** loop takes $O(|A||B|) = O(k^4)$ time. ◀

▶ Remark 13. One can modify Algorithm 2 by computing set intersections of the sets $N^{out}_{D/2,S}(a)$, $N^{in}_{\lceil D/2 \rceil,S}(b)$ for $a \in A, b \in B$, by brute force in lieu of matrix multiplication. This gives a combinatorial version of Algorithm 2 which runs in time $\tilde{O}(p^2 k) = \tilde{O}(k^5)$.

Next we give the directional min-distance tester. A call to this subroutine will prove a min-distance bound of $3D/2$ from $A$ to everything past $B$, prove a bound of $\lceil 3D/2 \rceil$ from $B$ to everything before $A$, or find a pair of vertices with min-distance greater than $D$. The idea is to iterate over vertices $a$ in $A$; those with large $D/2$-neighborhoods get hit by $S$, and if some $a$ has a small $D/2$-neighborhood it can be used as a jumping-off set to show $B$ is close to everything past $A$.

**Algorithm 2** All-pairs min-distance tester.

**Input:** DAG $G = (V, E)$, topological ordering $\pi$, subsets $A <_\pi B \subseteq V$ with $|A| = |B| = p = O(k^2)$, diameter guess $D$, $(k, (D/2, \lceil D/2 \rceil))$-neighborhood cover $S \subseteq V$ such that $\epsilon(s) \leq D$ for all $s \in S$, and $N^{out}_{D/2,S}(a)$, $N^{in}_{\lceil D/2 \rceil,S}(b)$ for $a \in A, b \in B$.

**Output:** One of the following. Each output verifies a corresponding property of $G$.

| | | |
|---|---|---|
| **PASS** | $\Rightarrow$ | $d_{\min}(a,b) \leq \lceil \frac{3}{2} D \rceil$ for all in $a \in A$ and $b \in B$ |
| **FAIL** | $\Rightarrow$ | `min-diameter`$(G) > D$ |

1 Compute $M_A$, the matrix with columns given by indicator vectors of $N^{out}_{D/2,S}(a)$ for $a \in A$;
2 Compute $M_B$, the matrix with columns given by indicator vectors $N^{in}_{\lceil D/2 \rceil,S}(b)$ for $b \in B$;
3 Compute $M = M_A^T M_B$;
4 **foreach** $a \in A, b \in B$ **do**
5  **if** $M_{ab} \geq 1$ **then**
6  **continue**;
7  **else if** $a <_\pi N^{in}_{\lceil D/2 \rceil,S}(b)$ **and** $N^{in}_{\lceil D/2 \rceil,S}(b) \cap S \neq \emptyset$ **then**
8  **continue**;
9  **else if** $b >_\pi N^{out}_{D/2,S}(a)$ **and** $N^{out}_{D/2,S}(a) \cap S \neq \emptyset$ **then**
10  **continue**;
11  **else**
12  **FAIL**;
13 **PASS**;

▶ **Lemma 14.** *Algorithm 3 produces the correct output in runtime $O(mk)$.*

**Proof.** We first show correctness. If the algorithm fails, we must have found a vertex $u$ with $\epsilon(u) > D$, and thus `min-diameter`$(G) > D$.

If the algorithm returns $A$, then for each $a \in A$, there is some $s \in S \cap N^{out}_{D/2,S}(a)$ with $s$ appearing in $B$ or to its left. Thus for all $u >_\pi B$, we have $d(a,u) \leq d(a,s) + d(s,u) \leq D/2 + D = 3D/2$.

Otherwise, the algorithm returns $B$. Then the condition in line 2 must hold for some $a \in A$. Let $u <_\pi A$ and $b \in B$. Since we do not fail in line 5, we must have $d(a,b) \leq D$. Let $x$ be a midpoint of the shortest path from $a$ to $b$, so $d(a,x) \leq D/2$ and $d(x,b) \leq \lceil D/2 \rceil$. Since $N^{out}_{D/2,S}(a)$ is either not cut off by hitting $S$ or is cut off after $B$, we have $x \in N^{out}_{D/2,S}(a)$. Finally, since we do not fail in line 5, we must have $d(u,x) \leq D$. Concluding, $d(u,b) \leq d(u,x) + d(x,b) \leq D + \lceil D/2 \rceil \leq \lceil 3D/2 \rceil$. As $u, b$ were arbitrary, this completes the case.

We conclude with runtime analysis. The outer **foreach** loop repeats until we have covered every vertex in $A$ or the condition in line 2 is satisfied. Checking this condition takes time at most $O(k)$ for a total of $O(|A|k) = O(mk)$. If the condition is satisfied, we perform $1 + |N^{out}_{D/2,S}(a)|$ calls to BFS, for a total of $O(mk)$. The algorithm concludes before returning to the outer loop, so we may add these contributions for a total time of $O(mk)$. ◀

We can now complete the analysis of the overall algorithm.

▶ **Lemma 15.** *Algorithm 1 produces the correct output in runtime $\tilde{O}(m^{\frac{4\beta+2-2\alpha\beta}{5\beta+3-2\alpha\beta}} n)$.*

**Algorithm 3** Directional min-distance tester.

**Input:** DAG $G = (V, E)$, topological ordering $\pi$, closed subsets $A <_\pi B \subseteq V$, diameter guess $D$, parameter $k$, $(k, (D/2, \lceil D/2 \rceil))$-neighborhood cover $S \subseteq V$ with $\epsilon(s) \leq D$ for all $s \in S$, and neighborhoods $N^{out}_{D/2,S}(a)$ for all $a \in A$.

**Output:** One of the following. Each output verifies a corresponding property of $G$.

| | | |
|---|---|---|
| **PASS** and return $A$ | $\Rightarrow$ | $d(a, v) \leq \frac{3}{2}D$ for all $a \in A$ and $v >_\pi B$ |
| **PASS** and return $B$ | $\Rightarrow$ | $d(v, b) \leq \lceil \frac{3}{2}D \rceil$ for all $b \in B$ and $v <_\pi A$ |
| **FAIL** | $\Rightarrow$ | $\texttt{min-diameter}(G) > D$ |

```
1 foreach a ∈ A do
2     if N^out_{D/2,S}(a) ∩ S = ∅ or N^out_{D/2,S}(a) ∩ S >_π B then
3         foreach v ∈ {a} ∪ N^out_{D/2,S}(a) do
4             BFS to and from v;
5             if ε(v) > D then
6                 FAIL;
7         PASS and return B;
8 PASS and return A;
```

**Proof.** This algorithm fails only when some $s \in S$ has $\epsilon(s) > D$, when Algorithm 2 fails, or Algorithm 3 fails, all of which imply $\texttt{min-diameter}(G) > D$. We now show that in the event of a pass, $\texttt{min-diameter}(G) \leq \lceil 3D/2 \rceil$. It suffices to prove that if the algorithm reaches line 13 then all min-distances between vertices in $L$ and $R$ are at most $\lceil 3D/2 \rceil$.

Assume we are at the beginning of iteration $t$ of the **while** loop. Let $I_t = \bigcup_{i<\ell<j} V_\ell$ be the interval strictly between $V_i$ and $V_j$. We will show inductively that for all $x \in L$ and $y \in R$, where at least one of $x$ or $y$ is in the interval $I_t$, $d_{\min}(x, y) \leq \lceil 3D/2 \rceil$. When the loop terminates, $I_t$ must entirely contain either $L$ or $R$, which will prove that all min-distance between vertices in $L$ and $R$ are at most $\lceil 3D/2 \rceil$.

The base case is trivial, as $I_1$ is empty. Assume the claim holds for $t$. Without loss of generality, assume Algorithm 3 returns $V_i$, so that $I_{t+1} = I_t \cup V_i$. Let $x \in L, y \in R$ with at least one of $x$ or $y$ in $I_{t+1}$. If $x$ or $y$ is in $I_t$, then $d(x, y) \leq \lceil 3D/2 \rceil$ by induction. Otherwise, one must lie in $I_{t+1} \setminus I_t = V_i$, and since $V_i \subset L$, this vertex must be $x$. Then we have $y \in V_j$ or $y >_\pi V_j$. If $y \in V_j$, then since Algorithm 2 did not fail, we have $d(x, y) \leq 3D/2$. If $y >_\pi V_j$, then because Algorithm 3 returned $V_i$ we have $d(x, y) \leq \lceil 3D/2 \rceil$. This completes the induction and the proof of correctness.

$G$ can be topologically sorted in time $\tilde{O}(n)$. Lemma 8 constructs the set $S$ in time $O(nk^2)$. Running BFS to and from each vertex in $S$ takes time $\tilde{O}(mn/k)$. We run Algorithm 2 and Algorithm 3 each up to $2\theta = n/2k^2$ times. Since Algorithm 2 takes time $O(k^{4+\frac{2\beta-2\alpha\beta}{\beta+1}})$ and Algorithm 3 takes time $O(mk)$, the total runtime of a recursive step is therefore:

$$\tilde{O}(mn/k + nk^{2+\frac{2\beta-2\alpha\beta}{\beta+1}})$$

Setting $k = m^{\frac{\beta+1}{5\beta+3-2\alpha\beta}}$, we obtain $\tilde{O}(m^{\frac{4\beta+2-2\alpha\beta}{5\beta+3-2\alpha\beta}}n)$ for each recursive step. The recursion over the left and right halves of the graph then adds a logarithmic factor. ◀

**Proof of Theorem 10.** Binary searching over $D \in [n]$ using Algorithm 1 gives the desired algorithm. The additive $+1/2$ follows from the fact that $\lceil 3D/2 \rceil$ may equal $3D/2 + 1/2$. ◀

▶ Remark 16. Using the combinatorial version of Algorithm 2, one can obtain a combinatorial $(\frac{3}{2}, \frac{1}{2})$-approximation algorithm for min-diameter which runs in time $\tilde{O}(m^{3/4}n)$.

▶ **Theorem 17.** *There is a $\frac{3}{2}$-approximation algorithm for min-diameter in unweighted DAGs that runs in time $\tilde{O}(m^{\frac{8\beta+4-4\alpha\beta}{5\beta+3-2\alpha\beta}} n^{\frac{\beta+1}{5\beta+3-2\alpha\beta}})$.*

This runtime is at most $O(m^{1.426}n^{0.288})$. We note that there is also a combinatorial version of this algorithm which runs in time $\tilde{O}(m^{5/4}n^{1/2})$, and a version which runs in $\tilde{O}(m^{\frac{7\beta+3-3\alpha\beta}{5\beta+3-2\alpha\beta}} n^{\frac{2\beta+2-\alpha\beta}{5\beta+3-2\alpha\beta}})$ time, which is $O(m^{1.171}n^{0.543})$, when $m \leq n^{1.283}$. A proof of the theorem may be found in the full version of the paper [7].

# 3 Bichromatic min-diameter

We first present OV-based conditional lower bounds for bichromatic min-diameter approximations, both for general graphs and for DAGs. Then in Section 3.2, we give an almost-2-approximation algorithm for bichromatic min-diameter in DAGs.

## 3.1 Conditional lower bounds for bichromatic min-diameter

We present, in order of increasing strength of the bound, three conditional lower bounds for approximating bichromatic min-diameter: one for DAGs, one for unweighted directed graphs, and one of weighted directed graphs. The constructions proceed analogously to Theorem 9, and can be found in the full version of the paper [7].

▶ **Theorem 18.** *For any $\epsilon, \delta > 0$ if there is an $O(n^{2-\epsilon})$-time algorithm giving a $(2-\delta)$-approximation for bichromatic min-diameter in unweighted DAGs with $O(n)$ vertices and $O(n^{1+o(1)})$ edges, then the OV Conjecture is false.*

▶ **Theorem 19.** *For any $\epsilon, \delta > 0$, if there is an $O(n^{2-\epsilon})$-time algorithm giving a $(5/2-\delta)$-approximation for bichromatic min-diameter in unweighted graphs with $O(n)$ vertices and $O(n^{1+o(1)})$ edges, then the OV Conjecture is false.*

▶ **Theorem 20.** *For any $\epsilon, \delta > 0$, if there is an $O(n^{2-\epsilon})$-time algorithm giving a $(3-\delta)$-approximation for bichromatic min-diameter in weighted graphs with $O(n)$ vertices and $O(n^{1+o(1)})$ edges, then the OV Conjecture is false.*

## 3.2 Almost-2-approximation for bichromatic min-diameter in DAGs

We conclude by turning to upper bounds for bichromatic min-diameter.

▶ **Theorem 21.** *There is an $\tilde{O}(\min(m^{4/3}n^{1/3}, m^{1/2}n^{3/2}))$-time algorithm, which, given a DAG $G$ with maximum red-blue edge weight $M_0$, outputs an approximation $D_0$ such that $D \leq D_0 < 2D + M_0 \leq 3D$.*

We give a brief overview of the algorithm. The full details and proof of correctness can be found in the full version of the paper [7].

We first consider the simpler case when the DAG is *separated*, i.e. when in some topological ordering every red vertex is to the left of every blue vertex. As is typical, we proceed by a binary search; at each stage we have some guess $D$ for the bichromatic min-diameter and either verify the true bichromatic min-diameter is larger than $D$ or smaller than $2D$.

We begin by finding a hitting set for the red vertices with large *red outneighborhoods* (other red vertices at distance at most $D$). This is achieved via the standard sampling method (Lemma 8). By running a BFS from this hitting set, we can verify min-distances for all vertices hit in this way to the blue side are not large; it remains to control vertices with small red out-neighborhoods. This is handled separately for sparse and dense graphs.

For sparse graphs, we add an additional step of BFS for each vertex of high degree. By sparsity, this must be a small fraction of the vertices, so this has a controllable contribution to the total runtime. For any red vertex with a high-degree vertex in its red outneighborhood, this will again verify an upper bound on the min-distance to any blue vertex. At this stage, if any vertex is left, its red outneighborhood is composed of few vertices of low degree. Its *blue boundary*, the set of blue neighbors of vertices in its red outneighborhood, must therefore be small. Consider a single such vertex. Any path from it to a blue vertex must intersect its blue boundary. So its blue boundary must be a hitting set for blue in-neighborhoods of all of the blue vertices, or our red vertex would be unable to reach all blue vertices in distance $D$. BFSing from this small blue hitting set completes the algorithm.

For dense graphs, the BFS steps from every vertex of high degree and every vertex in a blue boundary become too costly. After eliminating vertices with large neighborhoods via the randomized hitting set, we compute blue in-neighborhoods for all remaining blue vertices and blue boundaries for all remaining red vertices, and conclude by running set intersection on these two lists of sets.

We then bootstrap this special case for separated DAGs into a complete proof, by recursively splitting the DAG into a "middle" (consisting of a separated DAG) along with left and right halves, and bounding min-distances between these different sections of the graph.

## References


1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391, 2016.

3 D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

4 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 522–539. SIAM, 2021.

5 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 267–280, 2018.

6 J. Basch, S. Khanna, and R. Motwani. On diameter verification and boolean matrix multiplication. *Report No. STAN-CS-95-1544, Department of Computer Science, Stanford University (1995)*, 1995.

7 Aaron Berger, Jenny Kaufmann, and Virginia Vassilevska Williams. Approximating min-diameter: Standard and bichromatic, 2023. `arXiv:2308.08674`.

8 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376, 2016.

9 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the exact complexity of evaluating quantified $k$-cnf. In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation – 5th International Symposium, IPEC 2010, Chennai, India, December*

*13-15, 2010. Proceedings*, volume 6478 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2010.

10 Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert Endre Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1041–1052, 2014.

11 Shiri Chechik and Tianyi Zhang. Constant approximation of min-distances in near-linear time. In *Proceedings – 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science, FOCS 2022*, Proceedings – Annual IEEE Symposium on Foundations of Computer Science, FOCS, pages 896–906. IEEE Computer Society, 2022. Publisher Copyright: © 2022 IEEE.; null ; Conference date: 31-10-2022 Through 03-11-2022. doi:10.1109/FOCS54457.2022.00089.

12 F. R. K. Chung. Diameters of graphs: Old problems and new results. *Congr. Numer.*, 60:295–317, 1987.

13 Mina Dalirrooyfard and Jenny Kaufmann. Approximation Algorithms for Min-Distance Problems in DAGs. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.60.

14 Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, and Nicole Wein. Tight approximation algorithms for bichromatic graph diameter and related problems. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 47:1–47:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

15 Mina Dalirrooyfard, Virginia Vassilevska Williams, Nikhil Vyas, Nicole Wein, Yinzhan Xu, and Yuancheng Yu. Approximation algorithms for min-distance problems. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

16 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1150–1162. SIAM, 2012.

17 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

18 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

19 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, SCG '06, pages 52–60, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1137856.1137866.

20 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 515–524, 2013.

21 Virginia Vassilevska Williams. Lecture notes in graph algorithms cs267 (hitting sets, APSP), October 2016. URL: http://theory.stanford.edu/~virgi/cs267/lecture5.pdf.

22 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

23 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

24 Raphael Yuster. Computing the diameter polynomially faster than APSP. *arXiv preprint*, 2010. arXiv:1011.6181.

# Space-Efficient Parameterized Algorithms on Graphs of Low Shrubdepth

**Benjamin Bergougnoux**
Institute of Informatics, University of Warsaw, Poland

**Vera Chekan**
Humboldt-Universität zu Berlin, Germany

**Robert Ganian**
Algorithms and Complexity Group, TU Wien, Austria

**Mamadou Moustapha Kanté**
Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, CNRS, Clermont-Ferrand, France

**Matthias Mnich**
Hamburg University of Technology, Institute for Algorithms and Complexity, Germany

**Sang-il Oum**
Discrete Mathematics Group, Institute for Basic Science (IBS), Daejeon, Korea
Department of Mathematical Sciences, KAIST, Daejeon, Korea

**Michał Pilipczuk**
Institute of Informatics, University of Warsaw, Poland

**Erik Jan van Leeuwen**
Dept. Information and Computing Sciences, Utrecht University, The Netherlands

## Abstract

Dynamic programming on various graph decompositions is one of the most fundamental techniques used in parameterized complexity. Unfortunately, even if we consider concepts as simple as path or tree decompositions, such dynamic programming uses space that is exponential in the decomposition's width, and there are good reasons to believe that this is necessary. However, it has been shown that in graphs of low treedepth it is possible to design algorithms which achieve polynomial space complexity without requiring worse time complexity than their counterparts working on tree decompositions of bounded width. Here, *treedepth* is a graph parameter that, intuitively speaking, takes into account both the depth and the width of a tree decomposition of the graph, rather than the width alone.

Motivated by the above, we consider graphs that admit clique expressions with bounded depth and label count, or equivalently, graphs of low shrubdepth. Here, shrubdepth is a bounded-depth analogue of cliquewidth, in the same way as treedepth is a bounded-depth analogue of treewidth. We show that also in this setting, bounding the depth of the decomposition is a deciding factor for improving the space complexity. More precisely, we prove that on $n$-vertex graphs equipped with a tree-model (a decomposition notion underlying shrubdepth) of depth $d$ and using $k$ labels,

- Independent Set can be solved in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ using $\mathcal{O}(dk^2 \log n)$ space;
- Max Cut can be solved in time $n^{\mathcal{O}(dk)}$ using $\mathcal{O}(dk \log n)$ space; and
- Dominating Set can be solved in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ using $n^{\mathcal{O}(1)}$ space via a randomized algorithm.

We also establish a lower bound, conditional on a certain assumption about the complexity of Longest Common Subsequence, which shows that at least in the case of Independent Set the exponent of the parametric factor in the time complexity has to grow with $d$ if one wishes to keep the space complexity polynomial.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Parameterized complexity, shrubdepth, space complexity, algebraic methods

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.18





**Related Version** *Full Version*: https://arxiv.org/abs/2307.01285 [5]

**Funding** *Vera Chekan*: Supported by the DFG Research Training Group 2434 "Facets of Complexity".
*Robert Ganian*: Project No. Y1329 of the Austrian Science Fund (FWF), WWTF Project ICT22-029.
*Mamadou Moustapha Kanté*: Supported by the French National Research Agency (ANR-18-CE40-0025-01 and ANR-20-CE48-0002).
*Sang-il Oum*: Supported by the Institute for Basic Science (IBS-R029-C1).
*Michał Pilipczuk*: This work is a part of project BOBR that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 948057).

**Acknowledgements** This work was initiated at the *Graph Decompositions: Small Width, Big Challenges* workshop held at the Lorentz Center in Leiden, The Netherlands, in 2022.

## 1 Introduction

**Treewidth and Treedepth.** Dynamic programming on graph decompositions is a fundamental method in the design of parameterized algorithms. Among various decomposition notions, *tree decompositions*, which underly the parameter *treewidth*, are perhaps the most widely used; see e.g. [9, 12] for an introduction. A tree decomposition of a graph $G$ of width $k$ provides a way to "sweep" $G$ while keeping track of at most $k+1$ "interface vertices" at a time. This can be used for dynamic programming: during the sweep, the algorithm maintains a set of representative partial solutions within the part already swept, one for each possible behavior of a partial solution on the interface vertices. Thus, the width of the decomposition is the key factor influencing the number of partial solutions that need to be stored.

In a vast majority of applications, this number of different partial solutions depends (at least) exponentially on the width $k$ of the decomposition, which often leads to time complexity of the form $f(k) \cdot n^{\mathcal{O}(1)}$ for an exponential function $f$. This should not be surprising, as most problems where this technique is used are NP-hard. Unfortunately, the space complexity – which often appears to be the true bottleneck in practice – is also exponential. There is a simple tradeoff trick, first observed by Lokshtanov et al. [29], which can often be used to reduce the space complexity to polynomial at the cost of increasing the time complexity. For instance, Independent Set can be solved in $2^k \cdot n^{\mathcal{O}(1)}$ time and using $2^k \cdot n^{\mathcal{O}(1)}$ space on an $n$-vertex graph equipped with a width-$k$ tree decomposition via dynamic programming [19]; combining this algorithm with a simple recursive Divide&Conquer scheme yields an algorithm with running time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ and space complexity $n^{\mathcal{O}(1)}$.

Allender et al. [2] and then Pilipczuk and Wrochna [35] studied the question whether the loss on the time complexity is necessary if one wants to achieve polynomial space complexity in the context of dynamic programming on tree decompositions. While the formal formulation of their results is somewhat technical and complicated, the take-away message is the following: there are good complexity-theoretical reasons to believe that even in the simpler setting of path decompositions, one cannot achieve algorithms with polynomial space complexity whose running times asymptotically match the running times of their exponential-space counterparts. We refer to the works [2, 35] for further details.

However, starting with the work of Fürer and Yu [20], a long line of advances [25, 31, 32, 35] showed that bounding the *depth*, rather than the width, of a decomposition leads to the possibility of designing algorithms that are both time- and space-efficient. To this end, we consider the *treedepth* of a graph $G$, which is the least possible depth of an *elimination forest*: a forest $F$ on the vertex set of $G$ such that every two vertices adjacent in $G$ are in the

ancestor/descendant relation in $F$. An elimination forest of depth $d$ can be regarded as a tree decomposition of depth $d$, and thus treedepth is the bounded-depth analogue of treewidth. As shown in [20, 25, 32, 35], for many classic problems, including 3-Coloring, Independent Set, Dominating Set, and Hamiltonicity, it is possible to design algorithms with running time $2^{\mathcal{O}(d)} \cdot n^{\mathcal{O}(1)}$ and polynomial space complexity, assuming the graph is supplied with an elimination forest of depth $d$. In certain cases, the space complexity can even be as low as $\mathcal{O}(d + \log n)$ or $\mathcal{O}(d \log n)$ [35]. Typically, the main idea is to reformulate the classic bottom-up dynamic programming approach so that it can be replaced by a simple top-down recursion. This reformulation is by no means easy – it often involves a highly non-trivial use of algebraic transforms or other tools of algebraic flavor, such as inclusion-exclusion branching.

**Cliquewidth and Shrubdepth.** In this work, we are interested in the parameter *cliquewidth* and its low-depth counterpart: *shrubdepth*. While treewidth applies only to sparse graphs, cliquewidth is a notion of tree-likeness suited for dense graphs as well. The decompositions underlying cliquewidth are called *clique expressions* [8]. A clique expression is a term operating over *$k$-labelled graphs* – graphs where every vertex is assigned one of $k$ labels – and the allowed operations are: (i) apply any renaming function to the labels; (ii) make a complete bipartite graph between two given labels; and (iii) take the disjoint union of two $k$-labelled graphs. Then the cliquewidth of $G$ is the least number of labels using which (some labelling of) $G$ can be constructed. Similarly to treewidth, dynamic programming over clique expressions can be used to solve a wide range of problems, in particular all problems expressible in $\mathsf{MSO}_1$ logic, in FPT time when parameterized by cliquewidth. Furthermore, while several problems involving edge selection or edge counting, such as Hamiltonicity or Max Cut, remain W[1]-hard under the cliquewidth parameterization [16, 17], standard dynamic programming still allows us to solve them in XP time. In this sense, clique-width can be seen as the "least restrictive" general-purpose graph parameter which allows for efficient dynamic programming algorithms where the decompositions can also be computed efficiently [18]. Nevertheless, since the cliquewidth of a graph is at least as large as its linear cliquewidth, which in turn is as large as its pathwidth, the lower bounds of Allender et al. [2] and of Pilipczuk and Wrochna [35] carry over to the cliquewidth setting. Hence, reducing the space complexity to polynomial requires a sacrifice in the time complexity.

Shrubdepth, introduced by Ganian et al. [23], is a variant of cliquewidth where we stipulate the decomposition to have bounded depth. This necessitates altering the set of operations used in clique expressions in order to allow taking disjoint unions of multiple graphs as a single operation. In this context, we call the decompositions used for shrubdepth *$(d, k)$-tree-models*, where $d$ stands for the depth and $k$ for the number of labels used; a formal definition is provided in Section 2. Shrubdepth appears to be a notion of depth that is sound from the model-theoretic perspective, is FPT-time computable [21], and has become an important concept in the logic-based theory of well-structured dense graphs [13, 14, 22, 23, 33, 34].

Since shrubdepth is a bounded-depth analogue of cliquewidth in the same way as treedepth is a bounded-depth analogue of treewidth, it is natural to ask whether for graphs from classes of bounded shrubdepth, or more concretely, for graphs admitting $(d, k)$-tree-models where both $d$ and $k$ are considered parameters, one can design space-efficient FPT algorithms. Exploring this question is the topic of this work.

**Our contribution.** We consider three example problems: Independent Set, Max Cut, and Dominating Set. For each of them we show that on graphs supplied with $(d, k)$-tree-models where $d = \mathcal{O}(1)$, one can design space-efficient fixed-parameter algorithms whose

running times asymptotically match the running times of their exponential-space counterparts working on general clique expressions. While we focus on the three problems mentioned above for concreteness, we in fact provide a more general algebraic framework, inspired by the work on the treedepth parameterization [20, 25, 31, 32, 35], that can be applied to a wider range of problems. Once the depth $d$ is not considered a constant, the running times of our algorithms increase with $d$. To mitigate this concern, we give a conditional lower bound showing that this is likely to be necessary if one wishes to keep the space complexity polynomial.

Recall that standard dynamic programming solves the Independent Set problem in time $2^k \cdot n^{\mathcal{O}(1)}$ and space $2^k \cdot n^{\mathcal{O}(1)}$ on a graph constructed by a clique expression of width $k$ [19]. Our first contribution is to show that on graphs with $(d, k)$-tree-models, the space complexity can be reduced to as low as $\mathcal{O}(dk^2 \cdot \log n)$ at the cost of allowing time complexity $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$. In fact, we tackle the more general problem of computing the independent set polynomial.

▶ **Theorem 1.1.** *There is an algorithm which takes as input an $n$-vertex graph $G$ along with a $(d, k)$-tree model of $G$, runs in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and uses at most $\mathcal{O}(dk^2 \log n)$ space, and computes the independent set polynomial of $G$.*

The idea of the proof of Theorem 1.1 is to reorganize the computation of the standard bottom-up dynamic programming by applying the zeta-transform to the computed tables. This allows a radical simplification of the way a dynamic programming table for a node is computed from the tables of its children, so that the whole dynamic programming can be replaced by top-down recursion. Applying just this yields an algorithm with space polynomial in $n$. We reduce space to $\mathcal{O}(dk^2 \log n)$ by computing the result modulo several small primes, and using space-efficient Chinese remaindering. This is inspired by the algorithm for Dominating Set on graphs of small treedepth of Pilipczuk and Wrochna [35].

In fact, the technique used to prove Theorem 1.1 is much more general and can be used to tackle all coloring-like problems of local character. We formalize those under a single umbrella by solving the problem of counting List $H$-homomorphisms (for an arbitrary but fixed pattern graph $H$), for which we provide an algorithm with the same complexity guarantees as those of Theorem 1.1. The concrete problems captured by this framework include, e.g., Odd Cycle Transveral and $q$-Coloring for a fixed constant $q$ (details in the full version).

Next, we turn our attention to the Max Cut problem. This problem is W[1]-hard when parameterized by cliquewidth, but it admits a simple $n^{\mathcal{O}(k)}$-time algorithm on $n$-vertex graphs provided with clique expressions of width $k$ [17]. Our second contribution is a space-efficient counterpart of this result for graphs equipped with bounded-depth tree-models.

▶ **Theorem 1.2.** *There is an algorithm which takes as input an $n$-vertex graph $G$ along with a $(d, k)$-tree model of $G$, runs in time $n^{\mathcal{O}(dk)}$ and uses at most $\mathcal{O}(dk \log n)$ space, and solves the Max Cut problem on $G$.*

Upon closer inspection, the standard dynamic programming for Max Cut on clique expressions solves a Subset Sum-like problem whenever aggregating the dynamic programming tables of children to compute the table of their parent. We apply the approach of Kane [27] that was used to solve Unary Subset Sum in logarithmic space: we encode the aforementioned Subset Sum-like problem as computing the product of polynomials, and use Chinese remaindering to compute this product in a space-efficient way.

Finally, we consider the Dominating Set problem, for which we prove the following.

▶ **Theorem 1.3.** *There is a randomized algorithm which takes as input an $n$-vertex graph $G$ along with a $(d, k)$-tree model of $G$, runs in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and uses at most $\mathcal{O}(dk^2 \log n + n \log n)$ space, and reports the minimum size of a dominating set in $G$ that is correct with probability at least $1/2$.*

Note that the algorithm of Theorem 1.3 is randomized and uses much more space than our previous algorithms: more than $n \log n$. The reason for this is that we use the inclusion-exclusion approach proposed very recently by Hegerfeld and Kratsch [26], which is able to count dominating sets only modulo 2. Consequently, while the parity of the number of dominating sets of certain size can be computed in space $\mathcal{O}(dk^2 \log n)$, to determine the existence of such dominating sets we use the Isolation Lemma and count the parity of the number of dominating sets of all possible weights. This introduces randomization and necessitates sampling – and storing – a weight function. At this point we do not know how to remove neither the randomization nor the super-linear space complexity in Theorem 1.3; we believe this is an excellent open problem.

Note that in all the algorithms presented above, the running times contain a factor $d$ in the exponent compared to the standard (exponential-space) dynamic programming on clique expressions. The following conditional lower bound shows that some additional dependency on the depth is indeed necessary; the relevant precise definitions are provided in Section 4.

▶ **Theorem 1.4.** *Suppose* Longest Common Subsequence *cannot be solved in time* $M^{f(r)}$ *and space* $f(r) \cdot M^{\mathcal{O}(1)}$ *for any computable function* $f$*, even if the length* $t$ *of the sought subsequence is bounded by* $\delta(N)$ *for any unbounded computable function* $\delta$*; here* $r$ *is the number of strings on input,* $N$ *is the common length of each string, and* $M$ *is the total bitsize of the instance. Then for every unbounded computable function* $\delta$*, there is no algorithm that solves the* Independent Set *problem in graphs supplied with* $(d, k)$*-tree-models satisfying* $d \leqslant \delta(k)$ *that would run in time* $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ *and simultaneously use* $n^{\mathcal{O}(1)}$ *space.*

The possibility of achieving time- and space-efficient algorithms for Longest Common Subsequence was also the base of conjectures formulated by Pilipczuk and Wrochna [35] for their lower bounds against time- and space-efficient algorithms on graphs of bounded pathwidth. The supposition made in Theorem 1.4 is a refined version of those conjectures that takes also the length of the sought subsequence into account. The reduction underlying Theorem 1.4 is loosely inspired by the constructions of [35], but requires new ideas due to the different setting of tree-models of low depth.

Finally, given that the above results point to a fundamental role of shrubdepth in terms of space complexity, it is natural to ask whether shrubdepth can also be used to obtain meaningful tractability results with respect to the "usual" notion of fixed-parameter tractability. We conclude our exposition by highlighting two examples of problems which are NP-hard on graphs of bounded cliquewidth (and even of bounded pathwidth) [7, 28], and yet which admit fixed-parameter algorithms when parameterized by the shrubdepth.

▶ **Theorem 1.5.** *Metric Dimension and Firefighter can be solved in fixed-parameter time on graphs supplied with* $(d, k)$*-tree-models, where* $d$ *and* $k$ *are considered the parameters.*

In this work some technical details have been omitted due to space constraints. We refer to the full version of the paper for all proofs [5].

## 2 Preliminaries

For a positive integer $k$, we denote by $[k] = \{1, \ldots, k\}$ and $[k]_0 = [k] \cup \{0\}$. For a function $f \colon A \to B$ and elements $a, b$ (not necessarily from $A \cup B$), the function $f[a \mapsto b] \colon A \cup \{a\} \to B \cup \{b\}$ is given by $f[a \mapsto b](x) = f(x)$ for $x \neq a$ and $f[a \mapsto b](a) = b$. We use standard graph terminology [11]. The full proofs of our results also require the use of algebraic tools – notably the cover product and the fast subset convolution machinery of Björklund et al. [6].

We use the same computational model as Pilipczuk and Wrochna [35], namely the RAM model where each operation takes time polynomially proportional to the number of bits of the input, and the space is measured in terms of bits. We say that an algorithm $A$ runs in time $t(n)$ and space $s(n)$ if, for every input of size $n$, the number of operations of $A$ is bounded by $t(n)$ and the auxiliary used space of $A$ has size bounded by $s(n)$ bits.

**Shrubdepth.** We first introduce the decomposition notion for shrubdepth: *tree-models*.

▶ **Definition 2.1.** *For $d, k \in \mathbb{N}$, a $(d,k)$-tree-model $(T, \mathcal{M}, \mathcal{R}, \lambda)$ of a graph $G$ is a rooted tree $T$ of depth $d$ together with a family of symmetric Boolean $k \times k$-matrices $\mathcal{M} = \{M_a\}_{a \in V(T)}$, a labeling function $\lambda\colon V(G) \to [k]$, and a family of renaming functions $\mathcal{R} = \{\rho_{ab}\}_{ab \in E(T)}$ with $\rho_{ab}\colon [k] \to [k]$ for all $ab \in E(T)$ such that:*

- *The leaves of $T$ are identified with vertices of $G$. For each node $a$ of $T$, we denote by $V_a \subseteq V(G)$ the leaves of $T$ that are descendants of $a$, and with $G_a = G[V_a]$ we denote the subgraph induced by these vertices.*
- *With each node $a$ of $T$ we associate a labeling function $\lambda_a : V_a \to [k]$ defined as follows. If $a$ is a leaf, then $\lambda_a(a) = \lambda(a)$. If $a$ is a non-leaf node, then for every child $b$ of $a$ and every vertex $v \in V_b$, we have $\lambda_a(v) = \rho_{ab}(\lambda_b(v))$.*
- *For every pair of vertices $(u, v)$ of $G$, let $a$ denote their least common ancestor in $T$. Then we have $uv \in E(G)$ if and only if $M_a[\lambda_a(u), \lambda_a(v)] = 1$.*

We introduce some notation. If $(T, \mathcal{M}, \mathcal{R}, \lambda)$ is a $(d,k)$-tree model of a graph $G$, then for every node $a$ of $T$ and every $i \in [k]$, let $V_a(i) = \lambda_a^{-1}(i)$ be the set of vertices labeled $i$ at $a$. Given a subset $X$ of $V_a$ and $i \in [k]$, let $X_a(i) = X \cap V_a(i)$ be the vertices of $X$ labeled $i$ at $a$.

We say that a class $\mathcal{C}$ of graphs has *shrubdepth $d$* if there exists $k \in \mathbb{N}$ such that every graph in $\mathcal{C}$ admits a $(d,k)$-tree-model. Thus, shrubdepth is a parameter of a graph class, rather than of a single graph; though there are functionally equivalent notions, such as *SC-depth* [23] or *rank-depth* [10], that are suited for the treatment of single graphs. We remark that in the original definition proposed by Ganian et al. [23], relabeling is not allowed; however, using either definition yields the same notion of shrubdepth. Moreover, throughout this work we abstract away from the computation of the tree-models themselves and assume that a $(d,k)$-tree-model of the considered graph is provided on input.

We note that a fixed-parameter algorithm for computing tree-models has been proposed by Gajarský and Kreutzer [21] (in the sense of Ganian et al. [23]). The approach of Gajarský and Kreutzer is essentially kernelization: they iteratively "peel off" isomorphic parts of the graph until the problem is reduced to a kernel of size bounded only in terms of $d$ and $k$. This kernel is then treated by any brute-force method. Consequently, a straightforward inspection of their algorithm [21] shows that it can be implemented with polynomial space; but not space of the form $(d+k)^{\mathcal{O}(1)} \cdot \log n$, due to the necessity of storing all the intermediate graphs in the kernelization process. We leave as an open question the computation of a $(d,k)$-tree model, for a given graph $G$, running in time $f(d,k) \cdot n^{\mathcal{O}(1)}$ and using space $(d+k)^{\mathcal{O}(1)} \cdot \log n$.

## 3 Space-Efficient Algorithms on Tree-Models

**Independent Set.** In this section, we provide a fixed-parameter algorithm computing the independent set polynomial of a graph in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and using $\mathsf{poly}(d,k) \log n$ space, when given a $(d,k)$-tree model. In particular, given a $(d,k)$-tree model $(T, \mathcal{M}, \mathcal{R}, \lambda)$ of an $n$-vertex graph $G$, our algorithm will allow to compute the number of independent sets of size $p$ for each $p \in [n]$. For simplicity of representation, we start by describing an algorithm that uses $\mathsf{poly}(d,k,n)$ space and then show how a result by Pilipczuk and Wrochna [35] can be applied to decrease the space complexity to $\mathsf{poly}(d,k) \log n$.

In order to simplify forthcoming definitions/statements, let $a$ be an internal node of $T$ with $b_1, \ldots, b_t$ as children. For $S \subseteq [k]$, we denote by $q(a, S, p)$ the number of independent sets $I$ of size $p$ of $G_a$ such that $S = \{i \in [k] : I_a(i) \neq \emptyset\}$. Let us define the polynomial $\mathsf{IS}(a, S) = \sum_{p \in \mathbb{N}} q(a, S, p) \cdot x^p$. For the root $r$ of $T$, the number of independent sets of $G$ of size $p$ is then given by $\sum_{S \subseteq [k]} q(r, S, p)$ and the independent set polynomial of $G$ is $\sum_{S \subseteq [k]} \mathsf{IS}(r, S)$. Therefore, the problem boils down to the computation of $\mathsf{IS}(r, S)$ and its coefficients $q(r, S, p)$. A usual way to obtain a polynomial or logarithmic space algorithm is a top-down traversal of a rooted tree-like representation of the input – in our case, this will be the tree model. In this top-down traversal, the computation of coefficients $q(a, S, p)$ of $\mathsf{IS}(a, S)$ makes some requests to the coefficients $q(b_i, S_i, p_i)$ of $\mathsf{IS}(b_i, S_i)$ for each $i \in [t]$, for some integer $p_i$, and some set $S_i$ of labels of $G_{b_i}$ so that $\sum_{i \in [t]} p_i = p$ and $\bigcup_{i \in [t]} \rho_{ab_i}(S_i) = S$. Since there are exponentially many (in $t$) possible partitions of $p$ into $t$ integers and $t$ can be $\Theta(n)$, we must avoid running over all such integer partitions, and this will be done by the fast computation of a certain subset cover.

We will later show that if some independent set of $G_a$ contains vertices of labels $i$ and $j$ with $M_a[i, j] = 1$, then all these vertices come from the same child of $a$. In particular, the vertices of label $i$ (rsp. $j$) cannot come from multiple children of $a$. To implement this observation, after fixing a set $S$ of labels, for each label class in $S$ we "guess" (i.e., branch on) whether it will come from a single child of $a$ or from many. Such a guess is denoted by $\alpha \colon S \to \{1_=, 2_\geqslant\}$. So, the assignment $\alpha$ will allow us to control the absence of edges in the sought-after independent set. For a fixed $\alpha$, naively branching over all possibilities of assigning the labels of $S$ to the children of $a$ with respect to $\alpha$ would take time exponential in $t$, which could be as large as $\Theta(n)$. We will use inclusion-exclusion branching to speed-up the computations while retaining the space complexity. In some sense, we will first allow less restricted assignments of labels to the children of $a$, and then filter out the ones that result in non-independent sets using the construction of a certain auxiliary graph. The former will be implemented by using "less restricted" guesses $\beta \colon S \to \{1_=, 1_\geqslant\}$ where $1_\geqslant$ reflects that vertices of the corresponding label come from at least one child of $a$. Note that if the vertices of some label $i$ come from exactly one child of $a$, then such an independent set satisfies both $\beta(i) = 1_=$ and $\beta(i) = 1_\geqslant$. Although it might seem counterintuitive, this type of guesses will enable a fast computation of a certain subset cover. After that, we will be able to compute the number of independent sets satisfying guesses of type $\alpha \colon S \to \{1_=, 2_\geqslant\}$ by observing that independent sets where some label $i$ occurs in at least two children of $a$ can be obtained by counting those where label $i$ occurs in at least one child and subtracting those where this label occurs in exactly one child.

We now proceed to a formalization of the above. Let $S \subseteq \lambda_a(V_a)$ and $\alpha \colon S \to \{1_=, 2_\geqslant\}$ be fixed. Let $s_1, \ldots, s_{|\alpha^{-1}(2_\geqslant)|}$ be an arbitrary linear ordering of $\alpha^{-1}(2_\geqslant)$. To compute the number of independent sets that match our choice of $\alpha$, we proceed by iterating over $c \in \{0, \ldots, |\alpha^{-1}(2_\geqslant)|\}$, and we count independent sets where the labels in $\{s_1, \ldots, s_c\}$ occur exactly once, and the number of such sets where the labels occur at least once. Later, we will obtain the desired number of independent sets via carefully subtracting these two values. In particular, let $\gamma \colon \{s_1, \ldots, s_c\} \to \{1_=, 1_\geqslant\}$, and we denote by $q(a, S, \alpha, c, \gamma, p)$ the number of independent sets $I$ of size $p$ of $G_a$ such that

- for every label $i \notin S$, we have $I_a(i) = \emptyset$;
- for every label $i \in \{s_1, \ldots, s_c\}$ with $\gamma(i) = 1_=$, there exists a unique child $b_j$ of $a$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$;
- for every label $i \in \{s_1, \ldots, s_c\}$ with $\gamma(i) = 1_\geqslant$, there exists at least one child $b_j$ of $a$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$;

- for every label $i \in S \setminus \{s_1, \ldots, s_c\}$ with $\alpha(i) = 1_=$, there exists a unique child $b_j$ of $a$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$;
- and for every label $i \in S \setminus \{s_1, \ldots, s_c\}$ with $\alpha(i) = 2_\geqslant$, there exist at least two children $b_{j_1}$ and $b_{j_2}$ of $a$ such that $I_a(i) \cap V_{b_{j_1}} \neq \emptyset$ and $I_a(i) \cap V_{b_{j_2}} \neq \emptyset$.

We now proceed with some observations that directly follow from the definitions.

▶ **Observation 3.1.** *We have* $q(a, S, p) = \sum_{\alpha \in \{1_=, 2_\geqslant\}^S, \gamma \in \{1_=, 1_\geqslant\}^\emptyset} q(a, S, \alpha, 0, \gamma, p)$ *for every* $S \subseteq \lambda_a(V_a)$ *and integer* $p$. *Also, for every* $\alpha \in \{1_=, 2_\geqslant\}^S$, *every* $c \in \{0, \ldots, |\alpha^{-1}(2_\geqslant)| - 1\}$ *and every* $\gamma \colon \{s_1, \ldots, s_c\} \to \{1_=, 1_\geqslant\}$, *we have* $q(a, S, \alpha, c, \gamma, p) = q(a, S, \alpha, c+1, \gamma[s_{c+1} \mapsto 1_\geqslant], p) - q(a, S, \alpha, c+1, \gamma[s_{c+1} \mapsto 1_=], p)$.

It remains then to show how to compute the value $q(a, S, \alpha, |\alpha^{-1}(2_\geqslant)|, \gamma, p)$ for every $\alpha \in \{1_=, 2_\geqslant\}^S$, every $\gamma \in \{1_=, 1_\geqslant\}^{\alpha^{-1}(2_\geqslant)}$, and every integer $p$. It is worth mentioning that if $\beta \colon S \to \{1_=, 1_\geqslant\}$ is such that $\beta^{-1}(1_=) = \alpha^{-1}(1_=) \cup \gamma^{-1}(1_=)$ and $\beta^{-1}(1_\geqslant) = \alpha^{-1}(2_\geqslant) \setminus \gamma^{-1}(1_=)$, then $q(a, S, \alpha, |\alpha^{-1}(1_\geqslant)|, \gamma, p)$ is exactly the number of independent sets $I$ of size $p$ of $G_a$ satisfying the following:

1. For every $i \in [k] \setminus S$, we have $I_a(i) = \emptyset$.
2. For every $i \in \beta^{-1}(1_=)$, there exists a unique index $j \in [t]$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$.
3. For every $i \in \beta^{-1}(1_\geqslant)$, there exists a (not necessarily unique) index $j \in [t]$ such that $I_a(i) \cap V_{b_j} \neq \emptyset$.

We will therefore write $q(a, S, \beta, p)$ instead of $q(a, S, \alpha, |\alpha^{-1}(1_\geqslant)|, \gamma, p)$ and we define the polynomial $\mathsf{TIS}(a, S, \beta) \in \mathbb{Z}[x]$ (where "T" stands for "transformed") as $\mathsf{TIS}(a, S, \beta) = \sum_{p \in \mathbb{N}} q(a, S, \beta, p) \cdot x^p$. Recall that because we are computing $\mathsf{IS}(a, S)$ and $\mathsf{TIS}(a, S, \beta)$ in a top-down manner, some queries for $\mathsf{IS}(b_i, S_i)$ will be made during the computation. Before continuing in the computation of $\mathsf{TIS}(a, S, \beta)$, let us first explain how to request the polynomials $\mathsf{IS}(b_j, S_j)$ from each child $b_j$ of $a$. If $a$ is not the root, let $a^*$ be its parent in $T$, and we use $\mathsf{PIS}(a, S)$ (where "P" stands for "parent") to denote the polynomial $\mathsf{PIS}(a, S) = \sum_{p \in \mathbb{N}_0} q^\rho(a, S, p) x^p$ where $q^\rho(a, S, p) = \sum_{D \subseteq \lambda_a(V_a) \colon \rho_{a^* a}(D) = S} q(a, D, p)$ is the number of independent sets of $G_a$ of size $p$ that contain a vertex with label $i \in [k]$ (i.e., $I_{a*}(i) \neq \emptyset$) if and only if $i \in S$ holds, **where the labels are treated with respect to** $\lambda_{a^*}$. Then it holds that $\mathsf{PIS}(a, S) = \sum_{D \subseteq \lambda_a(V_a) \colon \rho_{a^* a}(D) = S} \mathsf{IS}(a, D)$ .

As our next step, we make some observations that will not only allow to restrict the $\beta$'s we will need in computing the polynomial $\mathsf{IS}(a, S)$ from the polynomials $\mathsf{TIS}(a, S, \beta)$, but will also motivate the forthcoming definitions. Recall that we have fixed $S \subseteq \lambda_a(V_a)$ and $\beta \colon S \to \{1_=, 1_\geqslant\}$, and in $\mathsf{IS}(a, S)$ and $\mathsf{TIS}(a, S, \alpha)$ we are only counting independent sets $I$ such that $I_a(i) \neq \emptyset$ if and only if $i \in S$.

▶ **Observation 3.2.** *If there exist* $i_1, i_2 \in S$ *such that* $M_a[i_1, i_2] = 1$, *then for any independent set* $I$ *counted in* $\mathsf{IS}(a, S)$, *there exists a unique* $j \in [t]$ *such that* $I_a(i_1) \cup I_a(i_2) \subseteq V_{b_j}$.

Recall that for every label $i \in \alpha^{-1}(2_\geqslant)$, each independent set $I$ contributing to the value $q(a, S, \alpha, 0, \gamma, p)$ has the property that there are distinct children $b_{j_1}$ and $b_{j_2}$ such that $I_a(i) \cap V_{b_{j_1}}$ and $I_a(i) \cap V_{b_{j_2}}$ are both non-empty. Then by Observation 3.2 for every $i_1 \in S$ it holds that if $\alpha(i_1) = 2_\geqslant$, then $M_a[i_1, i_2] = 0$ for all $i_2 \in S$. So if $\alpha$ does not satisfy this, the request $T(a, S, \alpha, 0, \gamma)$ can be directly answered with 0. Otherwise, since we use Observation 3.1 for recursive requests, the requests $\mathsf{TIS}(a, S, \beta)$ made all have the property that for each $i_1 \in S$ the following holds: if $\beta(i_1) = 1_\geqslant$, then $M_a[i_1, i_2] = 0$ for all $i_2 \in S$. We call such $\beta$'s *conflict-free* and we restrict ourselves to only conflict-free $\beta$'s. In other words, we may assume that if $i_1, i_2 \in S$ and $M_a[i_1, i_2] = 1$, then we have $\beta(i_1) = \beta(i_2) = 1_=$. Observation 3.2 implies that for such $i_1$ and $i_2$, each independent set $I$ counted in $\mathsf{TIS}(a, S, \beta)$

is such that $I_a(i_1) \cup I_a(i_2) \subseteq V_{b_j}$ for some child $b_j$ of $a$. Now, to capture this observation, we define an auxiliary graph $F^{a,\beta}$ as follows. The vertex set of $F^{a,\beta}$ is $\beta^{-1}(1_=)$ and there is an edge between vertices $i_1 \neq i_2$ if and only if $M_a[i_1, i_2] = 1$. Thus, by the above observation, if we consider a connected component $C$ of $F^{a,\beta}$, then in each independent set $I$ counted in $\mathsf{TIS}(a, S, \beta)$, all the vertices of $I$ with labels from $C$ come from a single child of $a$.

▶ **Observation 3.3.** *Let $C$ be a connected component of $F^{a,\beta}$. For every independent set $I$ counted in $\mathsf{TIS}(a, S, \beta)$, there exists a unique $j \in [t]$ such that $\bigcup_{i \in C} I_a(i) \subseteq V_{b_j}$.*

We proceed with some intuition on how we compute $\mathsf{TIS}(a, S, \beta)$ by requesting some $\mathsf{PIS}(b_j, S_j)$. Let $I$ be some independent set counted in $\mathsf{TIS}(a, S, \beta)$. This set contains vertices with labels from the set $S$, and the assignment $\beta$ determines whether there is exactly one or at least one child from which the vertices of a certain label come from. Moreover, by Observation 3.3, for two labels $i_1, i_2$ from the same connected component of $F^{a,\beta}$, the vertices with labels $i_1$ and $i_2$ in $I$ come from the same child of $a$. Hence, to count such independent sets, we have to consider all ways to assign labels from $S$ to subsets of children of $a$ such that the above properties are satisfied – namely, each connected component of $F^{a,\beta}$ is assigned to exactly one child while every label from $\beta^{-1}(1_\geqslant)$ is assigned to at least one child. Since the number of such assignments can be exponential in $n$, we employ the fast computation of a certain subset cover.

We now formalize this step. Let $\mathsf{cc}(F^{a,\beta})$ we denote the set of connected components of $F^{a,\beta}$. The universe $U^{a,\beta}$ (i.e., the set of objects we assign to the children of $a$) is defined as $U^{a,\beta} = \beta^{-1}(1_\geqslant) \cup \mathsf{cc}(F^{a,\beta})$. For every $j \in [t]$, we define a mapping $f_j^{a,\beta} \colon 2^{U^{a,\beta}} \to \mathbb{Z}[x, z]$ (i.e., to polynomials over $x$ and $z$) as follows: $f_j^{a,\beta}(X) = \mathsf{PIS}(b_j, \mathsf{flat}^{a,\beta}(X)) z^{|X \cap \mathsf{cc}(F^{a,\beta})|}$ where $\mathsf{flat}^{a,\beta} \colon 2^{U^{a,\beta}} \to 2^S$ intuitively performs a union over all the present labels – formally: $\mathsf{flat}^{a,\beta}(W) = (W \cap \beta^{-1}(1_\geqslant)) \cup \bigcup_{w \in W \cap \mathsf{cc}(F^{a,\beta})} w$. So if we fix the set $X$ of labels coming from the child $b_j$, then the (unique) coefficient in $f_j^{a,\beta}(X)$ reflects the number of independent sets of $G_{b_j}$ using exactly these labels (with respect to $\lambda_a$). The exponent of the formal variable $z$ is intended to store the number of connected components of $F^{a,\beta}$ assigned to $b_j$. This will later allow us to exclude from the computation those assignments of labels from $S$ to children of $a$ where the elements of some connected component of $F^{a,\beta}$ are assigned to multiple children of $a$. For every $j \in [t]$, we define a similar function $g_j^{a,\beta} \colon 2^S \to \mathbb{Z}[x, z]$ as follows:

$$g_j^{a,\beta}(Y) = \begin{cases} f_j^{a,\beta}(X) & \text{if } \mathsf{flat}^{a,\beta}(X) = Y \text{ for some } X \in 2^{U^{a,\beta}}, \\ 0 & \text{otherwise.} \end{cases}$$

Observe that the function $\mathsf{flat}^{a,\beta}$ is injective and hence $g_j^{a,\beta}$ is well-defined. The mapping $g_j^{a,\beta}$ filters out those assignments where some connected component of $F^{a,\beta}$ is "split".

▶ **Lemma 3.4.** *Let $(T, \mathcal{M}, \mathcal{R}, \lambda)$ be a $(d, k)$-tree model of an $n$-vertex graph $G$. Let $a$ be a non-leaf node of $T$ and let $b_1, \ldots, b_t$ be the children of $a$. For every $S \subseteq \lambda_a(V_a)$, and every conflict-free $\beta \colon S \to \{1_=, 1_\geqslant\}$, it holds that*

$$\mathsf{TIS}(a, S, \beta) = \left( \sum_{\substack{X_1, \ldots, X_t \subseteq [k] \colon \\ X_1 \cup \cdots \cup X_t = S}} \left( \prod_{j=1}^{t} g_j^{a,\beta}(X_j) \right) \right) \langle z^{|\mathsf{cc}(F^{a,\beta})|} \rangle,$$

*where for a polynomial $P = \sum_{u_1, u_2 \in \mathbb{N}_0} q_{u_1,u_2} x^{u_1} z^{u_2} \in \mathbb{Z}[x, z]$ the polynomial $P\langle z^{|\mathsf{cc}(F^{a,\beta})|} \rangle \in \mathbb{Z}[x]$ is defined as $P\langle z^{|\mathsf{cc}(F^{a,\beta})|} \rangle = \sum_{u_1 \in \mathbb{N}_0} q_{u_1, |\mathsf{cc}(F^{a,\beta})|} x^{u_1}$.*

Now we can apply a result by Björklund et al. [6] to accelerate the computation of $\mathsf{TIS}(a,S,\beta)$: It holds $\mathsf{TIS}(a,S,\beta) = \left(\sum_{Y\subseteq S}(-1)^{|S\setminus Y|}\prod_{j=1}^{t}\sum_{Z\subseteq Y} g_j(Z)\right)\langle z^{|\mathsf{cc}(F)|}\rangle$. We now have the equalities required for our algorithm to solve Independent Set parameterized by shrubdepth. By using these equalities directly, we would obtain an algorithm running in time $2^{\mathcal{O}(kd)}\cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2n^2)$. However, the latter can be substantially improved by using a result of Pilipczuk and Wrochna [35] based on the Chinese remainder theorem:

▶ **Theorem 3.5** ([35]). *Let $P(x)=\sum_{i=0}^{n'} q_i x^i$ be a polynomial in one variable $x$ of degree at most $n'$ with integer coefficients satisfying $0\leqslant q_i\leqslant 2^{n'}$ for $i=0,\ldots,n'$. Suppose that given a prime number $p\leqslant 2n'+2$ and $s\in\mathbb{F}_p$, the value $P(s) \pmod p$ can be computed in time $T$ and space $S$. Then given $k\in\{0,...,n'\}$, the value $q_k$ can be computed in time $\mathcal{O}(T\cdot\mathsf{poly}(n'))$ and space $\mathcal{O}(S+\log n')$.*

With this, we can finally prove Theorem 1.1.

**Counting List-Homomorphisms.** We now explain how to apply the techniques from the above to a broader class of problems, namely all problems expressible as instantiations of the #-List-$H$-Homomorphism problem for a fixed pattern graph $H$ (which we will introduce in a moment). In this way, we cover problems such as Odd Cycle Transversal and $q$-Coloring, for a fixed $q$. Furthermore, the techniques will be useful for solving Dominating Set later.

Let $H$ be a fixed undirected graph (possibly with loops) and let $R\subseteq V(H)$ be a designated set of vertices. An instance of the $R$-Weighted #-List-$H$-Homomorphism problem consists of a graph $G$, a weight function $\omega\colon V(G)\to\mathbb{N}$, a list function $L\colon V(G)\to 2^{V(H)}$, a *cardinality* $C\in\mathbb{N}$ and a *total weight* $W\in\mathbb{N}$. The goal is to count the number of list $H$-homomorphisms of $G$ such that exactly $C$ vertices of $G$ are mapped to $R$ and their total weight in $\omega$ is $W$. More formally, we seek the value

$$\Big|\big\{\varphi\colon V(G)\to V(H)\ \Big|\ \forall v\in V(G)\colon\varphi(v)\in L(v),\forall uv\in E(G)\colon\varphi(u)\varphi(v)\in E(H),\\ |\varphi^{-1}(R)|=C,\text{ and }\omega(\varphi^{-1}(R))=W\big\}\Big|\ .$$

We say that such $\varphi$ has cardinality $C$ and weight $W$. For the "standard" #-List $H$-Homomorphism problem we would use $R=V(H)$, $C=W=|V(G)|$, and unit weights. We also have the following special cases of the $R$-Weighted #-List-$H$-Homomorphism problem. In all cases, we consider unit weights.

- To model Independent Set, the pattern graph $H$ consists of two vertices $\mathbf{u}$ and $\mathbf{v}$ and the edge set contains a loop at $\mathbf{v}$ and the edge $\mathbf{uv}$. The set $R$ consists of $\mathbf{u}$ only.
- Similarly, to model Odd Cycle Transversal, the pattern graph $H$ is a triangle on vertex set $\{\mathbf{u},\mathbf{v},\mathbf{w}\}$ with a loop added on $\mathbf{u}$. Again, we take $R=\{\mathbf{u}\}$.
- To model $q$-Coloring, we take $H$ to be the loopless clique on $q$ vertices, and $R=V(H)$.

While in all the cases described above we only use unit weights, we need to work with any weight function in our application to Dominating Set. We prove the following result.

▶ **Theorem 3.6.** *Fix a graph $H$ (possibly with loops) and $R\subseteq V(H)$. There is an algorithm which takes as input an $n$-vertex graph $G$ together with a weight function $\omega$ and a $(d,k)$-tree-model, runs in time $2^{\mathcal{O}(dk)}\cdot n^{\mathcal{O}(1)}\cdot(W^*)^{\mathcal{O}(1)}$ and uses space $\mathcal{O}(k^2d(\log n+\log W^*))$, and solves the $R$-Weighted #-List-$H$-Homomorphism in $G$, where $W^*$ denotes the maximum weight in $\omega$.*

**Max Cut.** In the classical Max Cut problem, we are given a graph $G$ and the task is to output $\max_{X \subseteq V(G)} |E(X, V(G) \setminus X)|$. Towards solving the problem, let us fix a graph $G$ and a $(d,k)$-tree model $(T, \mathcal{M}, \mathcal{R}, \lambda)$ of $G$. Recall that for every node $a$ of $T$, $i \in [k]$ and $X \subseteq V_a$, we denote by $X_a(i)$ the set of vertices in $X$ labeled $i$ at $a$, i.e., $X \cap \lambda_a^{-1}(i)$. Given a child $b$ of $a$, we let $V_{ab} = V_b$ and we denote by $V_{ab}(i)$ the set of vertices in $V_b$ labeled $i$ at $a$, *i.e.*, $V_b \cap V_a(i)$. By $X_{ab}(i)$ we denote the set $X \cap V_{ab}(i)$. Given $c \in \{a, ab\}$, we define the *c-signature* of $X \subseteq V_c$ – denoted by $\mathsf{sig}_c(X)$ – as the vector $(|X_c(1)|, |X_c(2)|, \ldots, |X_c(k)|)$. We let $\mathcal{S}(c)$ be the set of $c$-signatures of all the subsets of $V_c$, *i.e.*, $\mathcal{S}(c) := \{\mathsf{sig}_c(X) : X \subseteq V_c\}$. Observe that $|\mathcal{S}(c)| \in n^{\mathcal{O}(k)}$ holds. Also, for the children $b_1, \ldots, b_t$ of $a$, we define $\mathcal{S}(ab_1, \ldots, ab_t)$ as the set of all tuples $(s^1, \ldots, s^t)$ with $s^i \in \mathcal{S}(ab_i)$ for each $i \in [t]$. Given $s \in \mathcal{S}(c)$, we define $f_c(s)$ as the maximum of $|E(X, V_c \setminus X)|$ over all the subsets $X \subseteq V_c$ with $c$-signature $s$. To solve Max Cut on $G$, it suffices to compute $\max_{s \in \mathcal{S}(r)} f_r(s)$ where $r$ is the root of $T$.

Let $b$ be a child of $a$. We start explaining how to compute $f_{ab}(s)$ by making at most $n^{\mathcal{O}(k)}$ calls to the function $f_b$. Given $s' \in \mathcal{S}(b)$, we define $\rho_{ab}(s')$ as the vector $s = (s_1, \ldots, s_k) \in \mathcal{S}(ab)$ such that, for each $i \in [k]$, we have $s_i = \sum_{j \in \rho_{ab}^{-1}(i)} s'_j$. Observe that for every $X \subseteq V_b$, we have $\mathsf{sig}_{ab}(X) = \rho_{ab}(\mathsf{sig}_b(X))$. Consequently, for every $s \in \mathcal{S}(ab)$, $f_{ab}(s)$ is the maximum of $f_b(s')$ over the $b$-signatures $s' \in \mathcal{S}(b)$ such that $\rho_{ab}(s') = s$. It follows that we can compute $f_{ab}(s)$ with at most $n^{\mathcal{O}(k)}$ calls to the function $f_b$.

▶ **Observation 3.7.** *Given a node $a$ of $T$ with a child $b$ and $s \in \mathcal{S}(ab)$, we can compute $f_{ab}$ in space $\mathcal{O}(k \log(n))$ and time $n^{\mathcal{O}(k)}$ with $n^{\mathcal{O}(k)}$ oracle access to the function $f_b$.*

In order to simplify forthcoming statements, we fix a node $a$ of $T$ with children $b_1, \ldots, b_t$. Now, we explain how to compute $f_a(s)$ by making at most $n^{\mathcal{O}(k)}$ calls to the functions $f_{ab_1}, \ldots, f_{ab_t}$. The first step is to express $f_a(s)$ in terms of $f_{ab_1}, \ldots, f_{ab_t}$. We first describe $|E(X, V_a \setminus X)|$ in terms of $|E(X \cap V_{b_i}, V_{b_i} \setminus X)|$. We denote by $E(V_{b_1}, \ldots, V_{b_t})$ the set of edges of $G[V_a]$ whose endpoints lie in different $V_{b_i}$'s, i.e. $E(G[V_{b_1}, \ldots, V_{b_t}]) \setminus (E(G[V_{b_1}] \cup \cdots \cup E(G[V_{b_t}]))$. Given $X \subseteq V_a$, we denote by $E_a(X)$ the intersection of $E(X, V_a \setminus X)$ and $E(V_{b_1}, \ldots, V_{b_t})$. In simple words, $E_a(X)$ is the set of all cut-edges (i.e., between $X$ and $V_a \setminus X$) running between distinct children of $a$. For $i, j \in [k]$, we denote by $E_a(X, i, j)$ the subset of $E_a(X)$ consisting of the edges whose endpoints are labeled $i$ and $j$. We capture the size of $E_a(X, i, j)$ with the following notion. For every $c \in \{a, ab_1, \ldots, ab_t\}$, $s \in \mathcal{S}(c)$ and $i, j \in [k]$, we define

$$\#\mathsf{pairs}_c(s, i, j) := \begin{cases} s_i \cdot (|V_c(j)| - s_j) + s_j \cdot (|V_c(i)| - s_i) \text{ if } i \neq j, \\ s_i \cdot (|V_c(i)| - s_i) \text{ otherwise.} \end{cases}$$

It is not hard to check that, for every subset $X \subseteq V_a$ with $a$-signature $s$, $\#\mathsf{pairs}_a(s, i, j)$ is the size of $\mathsf{pairs}_a(X, i, j)$ being the set of pairs of distinct vertices in $V_a$ labeled $i$ and $j$ at $a$ such that exactly one of them is in $X$. Observe that when $M_a[i, j] = 1$, then $|E_a(X, i, j)|$ is the number of pairs in $\mathsf{pairs}_a(X, i, j)$ whose endpoints belong to different sets among $V_{b_1}, \ldots, V_{b_t}$. Moreover, given a child $b$ of $a$, the number of pairs in $\mathsf{pairs}_a(X, i, j)$ whose both endpoints belong to $V_b$ is exactly $\#\mathsf{pairs}_{ab}(\mathsf{sig}_{ab}(X), i, j)$. Thus when $M_a[i, j] = 1$, we have

$$|E_a(X, i, j)| = \#\mathsf{pairs}_a(\mathsf{sig}_a(X), i, j) - \sum_{i \in [t]} \#\mathsf{pairs}_{ab_i}(\mathsf{sig}_{ab_i}(X), i, j) \ . \tag{1}$$

We capture the size of $E_a(X)$ with the following notion. For every $c \in \{a, ab_1, \ldots, ab_t\}$, $s \in \mathcal{S}(c)$ and $(k \times k)$-matrix $M$, we define $m_c(s, M) := \sum_{\substack{i,j \in [k], i \leqslant j \\ M[i,j]=1}} \#\mathsf{pairs}_c(s, i, j)$. Note that $|E_a(X)| = \sum_{i,j \in [k] :\, i \leqslant j, M_a[i,j]=1} |E_a(X, i, j)|$. Hence, by Equation 1, we deduce that $|E_a(X)| = m_a(\mathsf{sig}_a(X), M_a) - \sum_{i \in [t]} m_{ab_i}(\mathsf{sig}_{ab_i}(X), M_a)$. Since $E(X, V_a \setminus X)$ is the disjoint union of $E_a(X)$ and the sets $E(X \cap V_{b_1}, V_{b_1} \setminus X), \ldots, E(X \cap V_{b_t}, V_{b_t} \setminus X)$ , we deduce:

▶ **Observation 3.8.** *For every $X \subseteq V_a$ we have*

$$|E(X, V_a \setminus X)| = m_a(\mathsf{sig}_a(X), M_a) + \sum_{i=1}^{t} \left( |E(X_i \cap V_{b_i}, V_{b_i} \setminus X_i)| - m_{ab_i}(\mathsf{sig}_{ab_i}(X_i), M_a) \right) .$$

We are ready to express $f_a(s)$ in terms of $f_{ab_1}, \ldots, f_{ab_t}$ and $m_a, m_{ab_1}, \ldots, m_{ab_t}$.

▶ **Lemma 3.9.** *For every $s \in \mathcal{S}(a)$, we have*

$$f_a(s) = m_a(s, M_a) + \max_{\substack{(s^1,\ldots,s^t) \in \mathcal{S}(ab_1,\ldots,ab_t) \\ s = s^1 + \cdots + s^t}} \left( \sum_{i=1}^{t} \left( f_{ab_i}(s^i) - m_{ab_i}(s^i, M_a) \right) \right) .$$

To compute $f_a(s)$ we use a twist of Kane's algorithm [27] for solving the $k$-DIMENSIONAL UNARY SUBSET SUM in Logspace.

**Dominating Set.** We now prove Theorem 1.3. Note that DOMINATING SET cannot be directly stated in terms of $H$-homomorphisms for roughly the following reason. For $H$-homomorphisms, the constraints are *universal*: every neighbor of a vertex with a certain state must have one of allowed states. For DOMINATING SET, there is an *existential* constraint: a vertex in state "dominated" must have at least one neighbor in the dominating set. Also, the state of a vertex might change from "undominated" to "dominated" during the algorithm. The techniques we used for $H$-homomorphisms cannot capture such properties.

The problem occurs for other parameters as well. One approach that circumvents the issue is informally called *inclusion-exclusion branching*, and was used by Pilipczuk and Wrochna [35] in the context of DOMINATING SET on graphs of low treedepth. Their dynamic programming uses the states *Taken* (i.e., in a dominating set), *Allowed* (i.e., possibly dominated), and *Forbidden* (i.e., not dominated). These states reflect that we are interested in vertex partitions into three groups such that there are no edges between *Taken* vertices and *Forbidden* vertices; these are constraints that can be modelled using $H$-homomorphisms for a three-vertex pattern graph $H$. Crucially, for a single vertex $v$, if we fix the states of the remaining vertices, the number of partitions in which $v$ is dominated is given by the number of partitions where $v$ is possibly dominated minus the number of partitions where it is not dominated, i.e., informally "*Dominated = Allowed - Forbidden*".

For technical reasons explained later, our algorithm uses the classic Isolation Lemma:

▶ **Theorem 3.10** (Isolation lemma, [30])**.** *Let $\mathcal{F} \subseteq 2^{[n]}$ be a non-empty set family over the universe $[n]$. For each $i \in [n]$, choose a weight $\omega(i) \in [2n]$ uniformly and independently at random. Then with probability at least $1/2$ there exists a unique set of minimum weight in $\mathcal{F}$.*

Consequently, we pick a weight function $\omega$ that assigns every vertex a weight from $1, \ldots, 2n$ uniformly and independently at random. Storing $\omega$ takes $\mathcal{O}(n \log n)$ space. The remainder of the algorithm uses only $\mathcal{O}(dk^2 \log n)$ space.

To implement the above idea, we let the graph $H$ have vertex set $\{\mathbf{T}, \mathbf{A}, \mathbf{F}\}$ standing for *Taken*, *Allowed*, and *Forbidden*. This graph $H$ has a loop at each vertex as well as the edges **TA** and **AF**. Further, let $R \coloneqq \{\mathbf{T}\}$. Following our approach for $H$-homomorphisms, for every set $S \subseteq \mathbf{States}$ with $\mathbf{States} \coloneqq \{(\mathbf{T}, 1), (\mathbf{F}, 1), \ldots, (\mathbf{T}, k), (\mathbf{F}, k)\}$, every cardinality $c \in [n]_0$, and every weight $w \in [2n^2]_0$, in time $2^{\mathcal{O}(dk)} \cdot n^{\mathcal{O}(1)}$ and space $\mathcal{O}(dk^2 \log n)$ (recall that here for the maximum weight $W^*$ we have $W^* \leqslant 2n$) we can compute the value $a_{S,c,w}$ being the number of ordered partitions $(\widehat{T}, \widehat{F}, \widehat{A})$ of $V(G)$ satisfying the following properties:

1. there are no edges between $\widehat{T}$ and $\widehat{F}$;
2. $|\widehat{T}| = c$ and $\omega(\widehat{T}) = w$; and
3. for every $i \in [k]$ and $I \in \{T, F\}$, we have $(\mathbf{I}, i) \in S$ iff $\widehat{I} \cap V(i) \neq \emptyset$.

Note that we do not care whether vertices of some label $i$ are mapped to $A$ or not.

After that, we aim to obtain the number of dominating sets of cardinality $c$ and weight $w$ from values $a_{S,c,w}$. For this we need to transform the "states" *Allowed* and *Forbidden* into *Dominated*. Above we have explained how this transformation works if we know the state of a single vertex. However, now the set $S$ only captures for every label $i$, which states occur on the vertices of label $i$. First, the vertices of this label might be mapped to different vertices of $H$. And even if we take the partitions where all vertices of label $i$ are possibly dominated and subtract the partitions where all these vertices are not dominated, then we obtain the partitions where *at least one vertex* with label $i$ is dominated. However, our goal is that *all vertices* of label $i$ are dominated. So the *Dominated = Allowed - Forbidden* equality is not directly applicable here.

Recently, Hegerfeld and Kratsch [26] showed that when working with label sets, this equality is in some sense still true modulo 2. On a high level, they show that if we fix a part $\widehat{T}$ of a partition satisfying the above properties, then any undominated vertex might be put to any of the sides $\widehat{A}$ and $\widehat{F}$. Thus, if $\widehat{T}$ is not a dominating set of $G$, then there is an even number of such partitions and they cancel out modulo 2.

We can apply the same transformation to obtain from $a_{S,c,w}$'s the number of dominating sets of size $c$ and weight $w$ modulo 2. Isolation lemma implies that with probability at least $1/2$ for some $w$ this number if non-zero if a dominating set of size $c$ exists.

▶ Question 3.11. Is there an algorithm for Dominating Set of $n$-vertex graphs provided with a $(d, k)$-tree-model that runs in time $2^{\mathcal{O}(kd)} \cdot n^{\mathcal{O}(1)}$ and uses $(d+k)^{\mathcal{O}(1)} \log n$ space?

## 4 The Lower Bound

In this section, we prove Theorem 1.4. This lower bound is based on a reasonable conjecture on the complexity of the problem Longest Common Subsequence (LCS).

An instance of LCS is a tuple $(N, t, \Sigma, s_1, \ldots, s_r)$ where $N$ and $t$ are positive integers, $\Sigma$ is an alphabet and $s_1, \ldots, s_r$ are $r$ strings over $\Sigma$ of length $N$. The goal is to decide whether there exists a string $s \in \Sigma^t$ of length $t$ appearing as a subsequence in each $s_i$. There is a standard dynamic programming algorithm for LCS that has time and space complexity $\mathcal{O}(N^r)$. Abboud et al. [1] proved that the existence of an algorithm with running time $\mathcal{O}(N^{r-\varepsilon})$ for any $\varepsilon > 0$ would contradict the Strong Exponential-Time Hypothesis. As observed by Elberfeld et al. [15], LCS parameterized by $r$ is complete for the class XNLP: parameterized problems solvable by a nondeterministic Turing machine using $f(k) \cdot n^{\mathcal{O}(1)}$ time and $f(k) \log n$ space, for a computable function $f$. The only known progress on the space complexity is due to Barsky et al. with an algorithm running in $\mathcal{O}(N^{r-1})$ space [3]. This motivated Pilipczuk and Wrochna to formulate the following conjecture [35].

▶ **Conjecture 4.1** ([35]). *There is no algorithm that solves the LCS problem in time $M^{f(r)}$ and using $f(r)M^{\mathcal{O}(1)}$ space for any computable function $f$, where $M$ is the total bitsize of the instance and $r$ is the number of input strings.*

Note that in particular, the existence of an algorithm with time and space complexity as in Conjecture 4.1 implies the existence of such algorithms for all problems in the class XNLP.

Our lower bound is based on the following stronger variant of Conjecture 4.1, in which we additionally assume that the sought substring is short.

▶ **Conjecture 4.2.** *For any unbounded and computable function $\delta$, Conjecture 4.1 holds even when $t \leqslant \delta(N)$.*

Let $(N, t, \Sigma, s_1, \ldots, s_r)$ be an instance of LCS. We assume, without loss of generality, that $N$ is a power of 2. We provide a reduction from $(N, t, \Sigma, s_1, \ldots, s_r)$ to an equivalent instance of Independent Set consisting of a graph $G$ with $(r + t + N)^{\mathcal{O}(1)}$ vertices which admits a $(d, k)$-tree-model where $d = \mathcal{O}(\log t)$ and $k = \mathcal{O}(r \log N)$. This implies Theorem 1.4 since for every unbounded and computable function $\delta$ there exists an unbounded and computable function $\delta'$ such that if $t \leqslant \delta'(N)$, then $d \leqslant \delta(k)$ for all sufficiently large $N, r \in \mathbb{N}$.

To outline the main idea of the reduction, let $s^\star$ be a potential common substring of $s_1, \ldots, s_r$ of length $t$. We use matchings to represent the binary encoding of the positions of the letters of $s^\star$ in each string.

For every string $s_p$ and $q \in [t]$, we define the **selection gadget** $\mathsf{S}_p^q$ which contains, for every $i \in [\log N]$, an edge called the *$i$-edge* of $\mathsf{S}_p^q$. One endpoint of this edge is called the *0-endpoint* and the other is called the *1-endpoint*; i.e., a selection gadget induces a matching on $\log N$ edges. This results in the following natural bijection between $[N]$ and the maximal independent sets of $\mathsf{S}_p^q$. For every $I \in [N]$, we denote by $\mathsf{S}_p^q | I$ the independent set that contains, for each $i \in [\log N]$, the $x$-endpoint of the $i$-edge of $\mathsf{S}_p^q$ where $x$ is the value of the $i$-th bit of the binary representation of $I - 1$ (we consider the first bit to be the most significant one and the $\log N$-th one the least significant). Then the vertices selected in $\mathsf{S}_p^q$ encode the position of the $q$-th letter of $s^\star$ in $s_p$.

We need to guarantee that the selected positions in the gadgets $\mathsf{S}_p^1, \ldots, \mathsf{S}_p^t$ are coherent, namely, for every $q \in [t]$, the position selected in $\mathsf{S}_p^q$ is strictly smaller than the one selected in $\mathsf{S}_p^{q+1}$. For this, we construct an **inferiority gadget** denoted by $\mathsf{Inf}(p, q)$ for every string $s_p$ and every $q \in [t-1]$. The idea behind it is to ensure that the only possibility for an independent set to contain at least $3 \log N$ vertices from $\mathsf{S}_p^q, \mathsf{S}_p^{q+1}$, and their inferiority gadget, is the following: there exist $I < J \in [N]$ such that the independent set contains $\mathsf{S}_p^q | I \cup \mathsf{S}_p^{q+1} | J$. The maximum solution size in the constructed instance of Independent Set – which is the sum of the independence number of each gadget – will guarantee that only such selections are possible. We refer to the full version of this paper for the construction of these inferiority gadgets and the arguments proving the following observation.

▶ **Observation 4.3.** *Let $p \in [r]$ and $q \in [t-1]$. The independence number of $\mathsf{Inf}(p, q)$ is $\log N$ and for every $I, J \in [N]$, we have $I < J$ iff there exists a set of $\log N$ vertices $S$ from $\mathsf{Inf}(p, q)$ such that the union of $S$, $\mathsf{S}_p^q | I$ and $\mathsf{S}_p^{q+1} | J$ induces an independent set.*

Next, we need to ensure that the $t$ positions chosen in $s_1, \ldots, s_r$ indeed correspond to a common subsequence, i.e., for every $q \in [t]$, the $q$-th chosen letter must be the same in every $s_1, \ldots, s_r$. For $p \in [r-1]$, let $\mathcal{M}_p$ denote the set of all ordered pairs $(I, J) \in [N]^2$ such that the $I$-th letter of $s_p$ and the $J$-th of $s_{p+1}$ are identical. For each $p \in [r-1]$ and $q \in [t]$, we create the *matching gadget* $\mathsf{Match}(p, q)$ as follows:

- For every pair $(I, J) \in \mathcal{M}_p$ and for each $p^\star \in \{p, p+1\}$, we create a copy $\mathsf{M}_{p^\star}^{p,q,I,J}$ of $\mathsf{S}_{p^\star}^q$ and for every $\ell \in [\log N]$ and $x \in \{0, 1\}$, we add an edge between the $x$-endpoint of the $\ell$-edge of $\mathsf{S}_{p^\star}^q$ and the $(1-x)$-endpoint of the $\ell$-edge of $\mathsf{M}_{p^\star}^{p,q,I,J}$.
- For every pair $(I, J) \in \mathcal{M}_p$, we add a new vertex $v_{p,I,J}^q$ adjacent to (1) all the vertices from $\mathsf{M}_p^{p,q,I,J}$ that are not in $\mathsf{M}_p^{p,q,I,J} | I$ and (2) all the vertices from $\mathsf{M}_{p+1}^{p,q,I,J}$ that are not in $\mathsf{M}_{p+1}^{p,q,I,J} | J$.

Finally, we turn $\{v^q_{p,I,J} : (I,J) \in \mathcal{M}_p\}$ into a clique. Observe that, for each $p^\star \in \{p, p+1\}$, an independent set $S$ contains $(|\mathcal{M}_p|+1)\log N$ vertices from $\mathsf{S}^q_{p^\star}$ and its copies $\mathsf{M}^{p,q,I,J}_{p^\star}$ if and only if there exists a value $I \in [N]$ such that $S$ contains $\mathsf{S}^q_{p^\star}|I$ and $\mathsf{M}^{p,q,I,J}_{p^\star}|I$ for each copy. This leads to the following observation.

▶ **Observation 4.4.** *Let $p \in [r-1]$ and $q \in [t]$. The independence number of $\mathsf{Match}(p,q)$ is $1 + 2 \cdot |\mathcal{M}_p| \cdot \log N$ and for every $I, J \in [N]$, we have $(I,J) \in \mathcal{M}_p$ iff there exists an independent set $S$ of $\mathsf{Match}(p,q)$ with $1 + 2|\mathcal{M}_p| \cdot \log N$ vertices such that the union of $S$, $\mathsf{S}^q_p|I$ and $\mathsf{S}^q_{p+1}|J$ is an independent set.*

This concludes the construction of the graph $G$. See Figure 1 below for an overview.

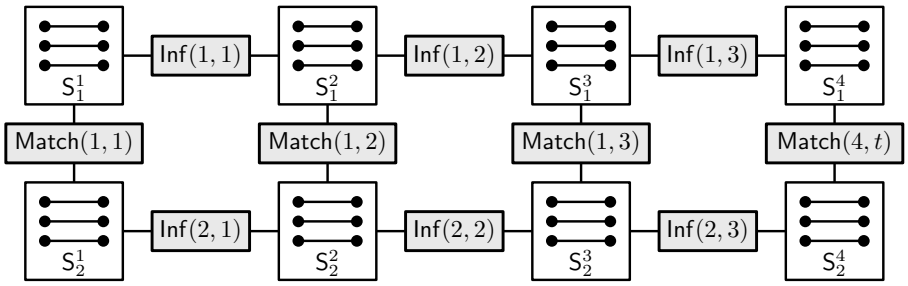



**Figure 1** Overview of the graph $G$ with $\log N = 3$, $r = 2$ and $t = 4$. There are some edges between two gadgets if and only if there are some edges between their vertices in $G$.

We prove correctness of the reduction in the following lemma which follows mostly from Observations 4.3 and 4.4.

▶ **Lemma 4.5.** *There exists an integer* goal *such that $G$ admits an independent set of size at least* goal *iff the strings $s_1, \ldots, s_r$ admit a common subsequence of length $t$.*

The next step is to construct a tree-model of $G$.

▶ **Lemma 4.6.** *We can compute in polynomial time a $(d,k)$-tree-model of $G$ where $d = 2\log t + 4$ and $k = 14r \log N - 3$.*

**Sketch of proof.** First, we prove that the union of the gadgets associated with a position $q \in [t]$ admits a simple tree-model. For every $q \in [t]$, we denote by $G^q$ the union of the selection gadgets $\mathsf{S}^q_p$ with $p \in [r]$ and the matching gadgets $\mathsf{Match}(p,q)$ with $p \in [r-1]$.

For each $q \in [t]$, we prove that $G^q$ admits a $(3,k)$-tree-model $(T^q, \mathcal{M}^q, \mathcal{R}^q, \lambda^q)$ where the tree $T^q$ is constructed as follows. We create the root $a^q$ of $T^q$ and we attach all the vertices in the selection gadgets $\mathsf{S}^q_p$ with $p \in [r]$ as leaves adjacent to $a^q$. Then, for every $p \in [r-1]$, we create a node $a^q_p$ adjacent to $a^q$ and for every $(I,J) \in \mathcal{M}_p$, we create a node $a^q_{p,I,J}$ adjacent to $a^q_p$. For each $(I,J) \in \mathcal{M}_p$, we make $a^q_{p,I,J}$ adjacent to the vertex $v^q_{p,I,J}$ and all the vertices in $\mathsf{M}^{p,q,I,J}_p$ and $\mathsf{M}^{p,q,I,J}_{p+1}$. Note that all the vertices in $\mathsf{Match}(p,q)$ are the leaves of the subtree rooted at $a^q_p$, and the leaves of $T^q$ are exactly the vertices in $G^q$. See Figure 2 for an illustration of $T^q$.

For every $q \in [t-1]$, we denote by $\mathsf{Inf}(q)$ the union of $\mathsf{Inf}(1,q), \ldots, \mathsf{Inf}(r,q)$. Moreover, for every interval $[x,y] \subseteq [t]$, we denote by $G^{x,y}$, the union of the graphs $G^q$ over $q \in [x,y]$, and the inferiority gadgets in $\mathsf{Inf}(q)$ over $q \in [x,y]$ such that $q+1 \in [x,y]$.

For every interval $[x,y]$, we prove by induction on $y-x$ that $G^{x,y}$ admits a $(2\log(y-x+1)+4, k)$-tree-model. In particular, it implies that $G^{1,t} = G$ admits a $(d,k)$-tree-model. It is also easy to see from our proof that this $(d,k)$-tree-model is computable in polynomial

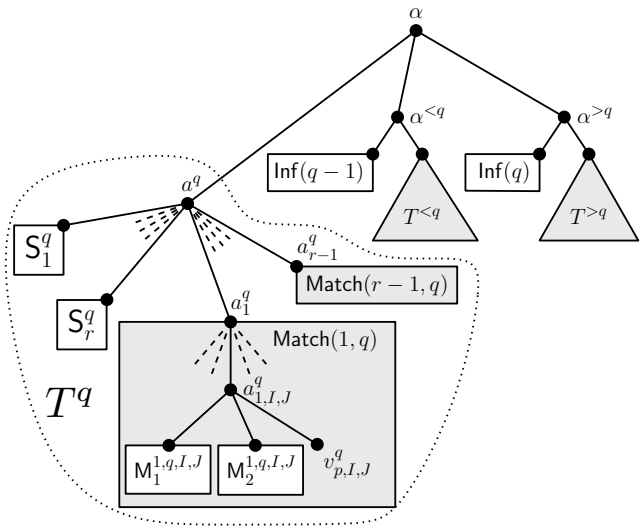



**Figure 2** Illustration of the tree $T$ and its subtree $T^q$ for the tree-model constructed in Lemma 4.6. An edge between a white filled rectangle labeled $\mathsf{X}$ and a node $a$ of the tree means that all the vertices in $\mathsf{X}$ are leaves adjacent to $a$.

time. For the base case of the induction, when $y = x$, we have $G^{x,y} = G^x$ and we have proved above that it admits a $(3, k)$-tree-model. When $x < y$, let $q = \lfloor (y - x)/2 \rfloor$. We use the induction hypothesis to obtain:

- A $(2\log(q - x) + 4, k)$-tree model $(T^{<q}, \mathcal{R}^{<q}, \mathcal{M}^{<q}, \lambda^{<q})$ for $G^{x,q-1}$.
- A $(2\log(y - q) + 4, k)$-tree-model $(T^{>q}, \mathcal{R}^{>q}, \mathcal{M}^{>q}, \lambda^{>q})$ for $G^{q+1,y}$.

Then, we construct a $(4 + 2\log(y - x + 1), k)$-tree-model $(T, \mathcal{R}, \mathcal{M}, \lambda)$ of $G^{x,y}$ from the tree-models of $G^{x,q-1}, G^{q+1,y}$, but also the $(3, k)$-tree-model $(T^q, \lambda^q, \mathcal{R}^q, \mathcal{M}^q)$ of $G^q$. To obtain $T$, we create the root $\alpha$ of $T$ and we make it adjacent to $a^q$, the root of $T^q$, and two new vertices: $\alpha^{<q}$ and $\alpha^{>q}$. We make $\alpha^{<q}$ adjacent to the root of $T^{<q}$ and to all the vertices in $\mathsf{Inf}(q-1)$. Symmetrically, we make $\alpha^{>q}$ adjacent to the root of $T^{>q}$ and to all the vertices in $\mathsf{Inf}(q)$. See Figure 2 for an illustration of $T$. ◀

## 5 Fixed-Parameter Algorithms for Metric Dimension and Firefighting

Theorem 1.5 – and in particular the fixed-parameter tractability of Metric Dimension and Firefighter parameterized by shrub-depth – can be obtained by combining known results about these problems [4, 24] with a bound on the maximum length of induced paths in graph classes of bounded shrubdepth [23, Theorem 3.7]. These results contrast the NP-hardness of both problems on graphs of bounded pathwidth [7, 28].

## References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proc. FOCS 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Eric Allender, Shiteng Chen, Tiancheng Lou, Periklis A. Papakonstantinou, and Bangsheng Tang. Width-parametrized SAT: Time–space tradeoffs. *Theory Comput.*, 10(12):297–339, 2014. doi:10.4086/toc.2014.v010a012.

3 Marina Barsky, Ulrike Stege, Alex Thomo, and Chris Upton. Shortest path approaches for the longest common subsequence of a set of strings. In *Proc. BIBE 2007*, pages 327–333, 2007. doi:10.1109/BIBE.2007.4375584.

4 Cristina Bazgan, Morgan Chopin, Marek Cygan, Michael R. Fellows, Fedor V. Fomin, and Erik Jan van Leeuwen. Parameterized complexity of firefighting. *J. Comput. System Sci.*, 80(7):1285–1297, 2014. doi:10.1016/j.jcss.2014.03.001.

5 Benjamin Bergougnoux, Vera Chekan, Robert Ganian, Mamadou Moustapha Kanté, Matthias Mnich, Sang il Oum, Michał Pilipczuk, and Erik Jan van Leeuwen. Space-efficient parameterized algorithms on graphs of low shrubdepth. *arXiv*, 2023. doi:10.48550/arXiv.2307.01285.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. STOC 2007*, pages 67–74, 2007.

7 Janka Chlebíková and Morgan Chopin. The firefighter problem: further steps in understanding its complexity. *Theoret. Comput. Sci.*, 676:42–51, 2017. doi:10.1016/j.tcs.2017.03.004.

8 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

9 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

10 Matt DeVos, O-joung Kwon, and Sang-il Oum. Branch-depth: Generalizing tree-depth of graphs. *European J. Combin.*, 90:Article 103186, 2020.

11 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 4th edition, 2012.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

13 Jan Dreier. Lacon- and shrub-decompositions: A new characterization of first-order transductions of bounded expansion classes. In *Proc. LICS 2021*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470680.

14 Jan Dreier, Jakub Gajarský, Sandra Kiefer, Michał Pilipczuk, and Szymon Toruńczyk. Treelike decompositions for transductions of sparse graphs. In *Proc. LICS 2022*, pages 31:1–31:14, 2022. doi:10.1145/3531130.3533349.

15 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.

16 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

17 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

18 Fedor V. Fomin and Tuukka Korhonen. Fast FPT-approximation of branchwidth. In *Proc. STOC 2022*, pages 886–899, 2022.

19 Martin Fürer. Multi-clique-width. In *Proc. ITCS 2017*, volume 67 of *Leibniz Int. Proc. Inform.*, pages 14:1–14:13, 2017. doi:10.4230/LIPIcs.ITCS.2017.14.

20 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory Comput. Syst.*, 61(2):283–304, 2017. doi:10.1007/s00224-017-9751-3.

21 Jakub Gajarský and Stephan Kreutzer. Computing shrub-depth decompositions. In *Proc. STACS 2020*, volume 154 of *Leibniz Int. Proc. Inform.*, pages 56:1–56:17, 2020. doi:10.4230/LIPIcs.STACS.2020.56.

22 Jakub Gajarský, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Trans. Comput. Log.*, 21(4):Art. 29, 41, 2020. doi:10.1145/3382093.

23 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Log. Methods Comput. Sci.*, 15(1):7:1–7:25, 2019. doi:10.23638/LMCS-15(1:7)2019.

24 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoret. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.

25 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In *Proc. STACS 2020*, volume 154 of *Leibniz Int. Proc. Inform.*, 2020.

26 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *arXiv*, 2023. doi:10.48550/ARXIV.2302.03627.

27 Daniel M. Kane. Unary subset-sum is in logspace. *arXiv*, 2010. arXiv:1012.1336.

28 Shaohua Li and Marcin Pilipczuk. Hardness of metric dimension in graphs of constant treewidth. *Algorithmica*, 84(11):3110–3155, 2022. doi:10.1007/s00453-022-01005-y.

29 Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar $k$-path in subexponential time and polynomial space. In *Proc. WG 2011*, volume 6986 of *Lecture Notes Comput. Sci.*, pages 262–270, 2011. doi:10.1007/978-3-642-25870-1_24.

30 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

31 Wojciech Nadara, Michał Pilipczuk, and Marcin Smulewicz. Computing treedepth in polynomial space and linear FPT time. In *Proc. ESA 2022*, volume 244 of *Leibniz Int. Proc. Inform.*, pages 79:1–79:14, 2022. doi:10.4230/lipics.esa.2022.79.

32 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In *Proc. WG 2020*, volume 12301 of *Lecture Notes Comput. Sci.*, pages 27–39, 2020. doi:10.1007/978-3-030-60440-0_3.

33 Pierre Ohlmann, Michał Pilipczuk, Wojciech Przybyszewski, and Szymon Toruńczyk. Canonical decompositions in monadically stable and bounded shrubdepth graph classes. *arXiv*, 2023. doi:10.48550/arXiv.2303.01473.

34 Patrice Ossona de Mendez, Michał Pilipczuk, and Sebastian Siebertz. Transducing paths in graph classes with unbounded shrubdepth. *European J. Combin.*, page 103660, 2022. doi:10.1016/j.ejc.2022.103660.

35 Michał Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

# High Performance Construction of RecSplit Based Minimal Perfect Hash Functions

**Dominik Bez**
Karlsruhe Institute of Technology, Germany

**Florian Kurpicz**
Karlsruhe Institute of Technology, Germany

**Hans-Peter Lehmann**
Karlsruhe Institute of Technology, Germany

**Peter Sanders**
Karlsruhe Institute of Technology, Germany

## Abstract

A minimal perfect hash function (MPHF) bijectively maps a set $S$ of objects to the first $|S|$ integers. It can be used as a building block in databases and data compression. RecSplit [Esposito et al., ALENEX'20] is currently the most space efficient practical minimal perfect hash function. It heavily relies on trying out hash functions in a brute force way.

We introduce *rotation fitting*, a new technique that makes the search more efficient by drastically reducing the number of tried hash functions. Additionally, we greatly improve the construction time of RecSplit by harnessing parallelism on the level of bits, vectors, cores, and GPUs.

In combination, the resulting improvements yield speedups up to 239 on an 8-core CPU and up to 5438 using a GPU. The original single-threaded RecSplit implementation needs 1.5 hours to construct an MPHF for 5 Million objects with 1.56 bits per object. On the GPU, we achieve the same space usage in just 5 seconds. Given that the speedups are larger than the increase in energy consumption, our implementation is more energy efficient than the original implementation.

**2012 ACM Subject Classification** Theory of computation → Data compression; Information systems → Point lookups

**Keywords and phrases** compressed data structure, parallel perfect hashing, bit parallelism, GPU, SIMD, parallel computing, vector instructions

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.19

**Related Version** *Extended Version*: `https://arxiv.org/abs/2212.09562` [5]

**Supplementary Material** *Software (Source Code)*: `https://github.com/ByteHamster/GpuRecSplit` archived at `swh:1:dir:1245e6eaef109ce4eb9f24080a2e9bdad7baf6d1`
*Software (Comparison with Competitors)*: `https://github.com/ByteHamster/MPHF-Experiments` archived at `swh:1:dir:890e76e03dd70e63eb57f3f62e466a4ee825cee4`

**Funding** This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

**Acknowledgements** This paper is based on and has text overlaps with Dominik Bez' Master's thesis [4]. We refer readers to that thesis for a detailed evaluation of the effects of low-level decisions like the choice of different similar SIMD instructions.

## 1 Introduction

A *Perfect Hash Function* (PHF) is a hash function that does not have collisions, i.e., is injective, on a given set $S$ of objects. Evaluating the PHF on any object not in $S$ can return an arbitrary value. A *Minimal Perfect Hash Function* (MPHF) maps the objects in $S$ to the first

$n = |S|$ integers, so it is bijective. MPHFs are useful in many applications, for example, to implement hash tables with guaranteed constant access time [21]. By storing only fingerprints in the hash table cells [3, 15], we obtain an *approximate membership data structure*. Storing payload data in the cells, we obtain an updatable retrieval data structure [32]. Finally, the perfect hash function values can be used as small identifiers of the input objects [6], which are easier to handle and more space efficient than, for example, strings.

MPHFs can be very compact – the theoretically minimal space usage is 1.44 bits per object [2]. Currently, the most space-efficient practical MPHF is RecSplit [14]. It provides various trade-offs between the space consumption, construction time, and query time.

In this paper, we provide several improvements inside the RecSplit framework. We first describe RecSplit and other preliminaries in Section 2 and briefly review related work in Section 3. As a core step during construction, RecSplit tries out hash functions on a small set of objects until one hash function is a bijection. We introduce a new bijection search mechanism in Section 4, which reduces the search space of the brute force algorithm compared to the original method. *Rotation fitting* hashes the objects to two sets and tries to fit one set into the "holes" of the other set by rotating (cyclically shifting) it. As a positive side effect, this approach makes good use of bit parallelism.

We then parallelize RecSplit (with and without rotation fitting) using the vector parallelism available with *Single Instruction Multiple Data* (SIMD) instructions and the thread parallelism available with multicore CPUs and GPUs. Given that hash function construction here is mostly compute bound and can be done in parallel for a huge number of small subproblems, the GPU is an ideal hardware. Utilizing GPUs for evaluating hash functions is known from mining of cryptocurrencies with proof-of-work approach (e.g., Bitcoin). Our extensive evaluation in Section 6 shows speedups of up to 50 using SIMD, 239 when additionally using multi-threading with 16 threads, and 5438 using a GPU, compared to the original single-threaded implementation without rotation fitting. Because GPUs are so much faster at constructing MPHFs, they lead to a better energy efficiency than the CPU, as we show in the experiments. Finally, in Section 7, we summarize the results and give directions for future research.

**Our Contributions.** With *rotation fitting*, we introduce a new method for searching for bijections that can be used in RecSplit. We significantly accelerate the construction by four kinds of parallelism (bits, vectors, multicores, and GPU). Together, this accelerates RecSplit constructions by a factor up to 5438 and even makes its construction performance competitive to significantly less space efficient minimal perfect hash functions.

## 2 Preliminaries

In Section 2.1, we first shortly describe basic techniques needed by our implementation. We then continue with describing RecSplit in detail in Section 2.2. Finally, we describe SIMD in Section 2.3 and GPUs in Section 2.4.

### 2.1 Basics

**Words and Bit Vectors.** An important operation in RecSplit is `popcount`, which returns the number of 1-bits in a word. Given a bit vector, the $rank_1(x)$ operation returns the number of 1-bits before position $x$, and the $select_1(x)$ operation returns the position of the $x$-th 1-bit. The operation can be executed in constant time [10, 27] and has very fast and space-efficient implementations [28, 39]. An additional operation we need in this paper is $\text{rot}_k^i(x)$ which rotates (i.e., cyclically shifts) the $k$ least significant bits of $x$ by $i$ bit positions. This can be implemented in a bit parallel way using shifting and masking.

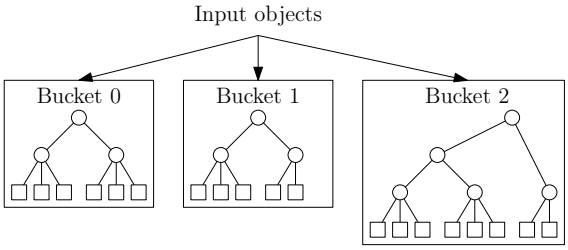



**Figure 1** Illustration of the overall RecSplit data structure. Circular nodes of the trees represent splittings, squares represent bijections.

**Golomb-Rice.** The Golomb code [24] is a variable length code that is optimal for geometric distributions. Golomb-Rice [38] is a faster special case, which is almost as space efficient. Given a parameter $\tau$ and the number $x$ to store, the $\tau$ least significant bits of $x$ form the *fixed part* which is stored directly. The remaining bits are encoded in unary, consisting of $\lfloor x/2^\tau \rfloor$ 0-bits and a final 1-bit. To access one element, we can get the lower bits from the array of fixed parts and the upper bits through two $select_1$ queries.

**Elias-Fano.** An Elias-Fano representation [13,16] can be used to store a monotonic sequence of integers $p_1, \ldots, p_k$ with $p_k \leq U$. Similar to Golomb-Rice codes, the least significant bits of each value are stored directly in the lower-bits array and can be accessed directly. The remaining most significant bits $u$ at index $i$ are encoded as a 1-bit in a bit vector at position $i + u$. This means that by executing a $select_1$ query on the upper bits and looking up the lower bits, we can restore any value in constant time. Using this representation, the sequence can be stored using $k(2 + \log(U/k))$ bits.

## 2.2 RecSplit

We now describe RecSplit [14], the MPHF that this paper is based on. Figure 1 illustrates the overall data structure. The first step of the construction is to apply an initial hash function on every object of the input to generate objects of uniform distribution. These objects are mapped to different buckets of expected size $b$, where $b$ is a tuning parameter.

**Splitting Trees.** In each bucket, RecSplit constructs an independent *splitting tree*. The tree partitions the objects into smaller and smaller sets until the individual sets have a small configurable size $\ell$. The splitting tree has a well-defined shape, depending only on the leaf size $\ell$ and the number of objects in the bucket. At each inner node, RecSplit tries random hash functions to find one that distributes the objects to the child nodes according to the tree structure. The number of child nodes of an inner node is called fanout. The fanout is optimized in such a way that the expected amount of work to find the splitting is roughly equal to the amount of work in all children combined. The fanouts of the two bottom-most levels are $\max\{2, \lceil 0.35\ell + 0.55 \rceil\}$ and $\max\{2, \lceil 0.21\ell + 0.9 \rceil\}$. In the terminology of the RecSplit paper, these levels are called *lower aggregation levels*. The levels above, also called *upper aggregation levels*, simply use a fanout of 2.

**Bijections.** The lowest level of the splitting tree is called *leaf level*. Each leaf, except for possibly the last, contains $\ell$ objects. This is small enough that it is feasible to search for a bijective mapping by trying random hash functions using brute force. The inner loop of the

bijection search applies a hash function modulo $\ell$ on each object. It converts the value to a bit by taking two to the power of it, and sets the corresponding bit in a bit vector of length $\ell$ using a logical `OR` operation. After hashing all objects, if the resulting bit vector has all its bits set to 1, it means that the hash function is a bijection on the leaf. If it is not, RecSplit tries the next hash function.

**Representation.** Because the splitting trees have a well-defined shape, it is enough to store the hash function identifier at each node in preorder. These numbers are encoded with Golomb-Rice code, where all unary parts and all binary parts of a tree are stored together. The optimal Golomb parameter $\tau$ is different based on the layer in the tree and can be pre-calculated. The encodings of all splitting trees from all buckets are concatenated in a single bit vector. An additional sequence with encoding based on Elias-Fano encodes both the prefix sums of the number of objects in each bucket and the positions where the encoding of each bucket starts.

**Query.** A RecSplit hash function can be evaluated by determining the bucket of an object and locating its encoding. The splitting tree in the bucket is traversed from the root to a leaf by applying the splitting hash function, which determines the child to descend into. Finding the encoding of a subtree is possible by executing a $select_1$ query on the upper bits of the Golomb-Rice coded hash function identifiers. During traversal, the number of objects stored in children left to the one descended into are accumulated. The final hash value is then the sum of the value of leaf bijection, the number of objects to the left in the splitting tree, and the total size of previous buckets.

The combination of brute force splitting and bijections is highly space efficient from an information-theoretical point of view – disregarding overheads due to encoding and metadata, optimal space consumption can be achieved. Consequently, as the leaf size $\ell$ gets larger, optimal space is approached [14].

## 2.3 SIMD

It is common, especially in perfect hashing, that the same operation needs to be executed on different data. This can be achieved with a simple loop, which means that the corresponding instructions must be decoded by the hardware for every element. This can be improved by using *Single Instruction, Multiple Data* (SIMD) [17]. A single instruction is used to apply the same operation on a *vector* of several elements. We refer to a single element within a SIMD vector as a *lane*. For example, a vector may contain 16 lanes with 32 bits each, i.e., the vector contains 512 bits overall. The exact set of operations depends on the concrete implementation of the SIMD model. On many Intel and AMD processors, SIMD operations are available through the Advanced Vector Extensions (AVX) [25]. AVX-512 [26] extends these operations to 512-bit vectors and is divided into many smaller subsets that offer additional operations. A subset that is useful for our implementation is AVX512VPOPCNTDQ, which provides `popcount` on 512-bit vectors with lanes of size 32 and 64 bits. The $\text{rot}_k^i$ function that cyclically shifts bits (see Section 2.1) can be implemented in parallel using SIMD.

## 2.4 GPUs

Graphics Processing Units (GPUs) are specialized processors initially designed for computer graphics applications. Over the last decades, GPUs evolved to general purpose processors for highly parallelizable tasks. We now describe the hardware and programming interface in the following paragraphs. To provide a grasp of the dimensions of a current GPU, we give metrics of the NVIDIA RTX 3090 [33], which is also used for our experiments (see Section 6).

**Compute Hardware.** A GPU consists of several streaming multiprocessors (SMs) (RTX 3090: 82). Each SM contains many arithmetic logic units (ALUs) to perform computations (RTX 3090: 64 integer ALUs). Several threads (RTX 3090: 32) operate in *lock-step*, i.e., they execute the same instruction at the same time. Such a bundle of threads is called *warp*. Threads are masked out for instructions they should not execute. This means that in loops, each thread in a warp has to iterate as many times as the thread with the largest number of iterations. To hide latencies, e.g., for memory access, each SM is oversubscribed with more threads than ALUs, and the GPU schedules the threads efficiently. Multiple warps of threads form a *thread block*. Thread blocks are guaranteed to reside on the same SM, which enables them to cooperate.

**Memory.** The *global memory* is the largest and slowest memory on the GPU (RTX 3090: 24 GB). When multiple threads of a warp access the memory simultaneously, the hardware serves the requests with as few memory transactions as possible. *Shared memory* is a fast memory placed on each SM. It is shared between the threads of the same thread block. On the RTX 3090, shared memory and L1 cache are allocated on the same memory areas. The data in shared memory is partitioned into 32 memory banks, and the $i$-th 32-bit word is stored in bank $i$ mod 32. When multiple threads simultaneously access different words within the same bank, the access operations have to be serialized.

**CUDA.** An efficient way to develop applications on NVIDIA GPUs is CUDA [34]. Functions which can be executed on the GPU are called *kernels*. Each kernel is executed on a *grid* of *thread blocks*. The grid size and the number of threads per block can be selected by the user. The user can create several *streams*. The kernels and data transfers launched into a specific stream are executed in order, but operations in different streams can arbitrarily overlap.

## 3 Related Work

Perfect Hashing is an active area of research [2, 7, 8, 9, 11, 20, 30, 31, 32, 37, 40]. Due to a lack of space, we only describe the most recent and fastest algorithms here. For a more detailed overview of recent methods, refer to Ref. [30]. To the best of our knowledge, there is no technique that constructs MPHFs on the GPU yet. Lefebvre and Hoppe [29] describe the GPU evaluation of MPHFs that were constructed on CPUs.

**FiPHa/BBHash.** A fast and simple approach to minimal perfect hashing uses fingerprinting and bumping [9, 31, 32]. BBHash [31] is a publicly available parallel implementation. The set $S$ of input objects is hashed using a hash function $h \to \beta n$ for a tuning parameter $\beta$. The set $S'$ of objects that have a collision is handled recursively. Consider the bit vector $b$ with $b[i] = 1$ iff $|\{s \in S : h(s) = i\}| = 1$. Then $rank_1(h(s))$ defines an MPHF on $S \setminus S'$. This approach needs at least $e$ bits per object (when $\beta = 1$) and provides efficient queries when about 4 or more bits per object are available (using larger values of $\beta$). An advantage is the very simple and easily parallelizable construction.

**PTHash.** PTHash [37] is based on FCH [20] which can be considered a predecessor of the hash-and-displace technique [2]. The objects are first distributed into different buckets using a hash function, but the distribution is not uniform. Specifically, about 60% of the objects are mapped to 30% of the buckets. The buckets are then processed in order of decreasing size. For each bucket, a hash function is searched such that each object can be placed in

the output domain without colliding with other objects that are already placed. The hash function identifiers are searched linearly and then stored in compressed form with several possible compression schemes. The proclaimed goal of PTHash is fast query times. Using an appropriate compression scheme, only a single memory access is required to find the hash value, and the remaining operations are simple hash function evaluations and arithmetic. Compared to the original implementation of RecSplit, PTHash consumes more space, but has faster queries and faster construction time. PTHash-HEM [36] is an implementation that first partitions the input and then constructs each partition independently in parallel.

**SicHash.** SicHash [30] is based on the simple idea to store the index of the hash function to be used in a retrieval data structure. It can capitalize on recent progress on fast and nearly space optimal retrieval [12]. Computing a valid index for all objects amounts to constructing a cuckoo hash table [19, 35]. In contrast to the brute force methods at the core of PTHash and RecSplit, this can be done in near linear time even on large tables. SicHash refines this basic approach using a mix of several fixed precision retrieval data structures and by using many small(ish) cuckoo hash tables rather than a single large table. Roughly, SicHash allows faster construction than PTHash while offering similar query time and space consumption.

## 4 Rotation Fitting

The general idea of RecSplit consists of two independent steps, bijections and splittings (see Section 2.2). In this section, we introduce a new method for searching for bijections in RecSplit's leaf nodes. As a reminder, given $m$ objects, we are looking for a way to quickly find a mapping of the objects to the first $m$ integers without any collisions. The original implementation tries out hash functions using brute force until one of them is a bijection.

*Rotation fitting* ensures that we need significantly fewer hash function evaluations. From the result of one evaluation, we derive additional candidates that are very fast to compute. Rotation fitting is efficient when $m \leq w$, where $w$ is the size of a machine word. We randomly distribute the objects into two sets $A$ and $B$ by using a 1-bit hash function. The 1-bit hash function is the same for all leaf nodes and does not ensure that $A$ and $B$ have the same size. Now we search for a hash function $h$ that gives a bijection on the leaf. Like in the original RecSplit implementation, we calculate the hash value of all objects in $A$ and set the respective bits in the word $a$ to 1. The function $h$ may be ruled out as a valid bijection by calculating the `popcount` of $a$. Analogously, the set $B$ is mapped to the word $b$ using the same hash function $h$. Let us now rotate (i.e., cyclically shift) the bits in $b$. If we can find a rotation value such that the 1-bits in $b$ fit exactly onto the 0-bits in $a$, we have found a bijection on the leaf. More formally, this is the case if there is an $r \in \{0, ..., m-1\}$, such that $a \,|\, \mathrm{rot}_m^r(b)$ has the $m$ least significant bits all set. In the extended version [5], we show that for large $m$ the probability of finding a bijection using rotation fitting is about $m$ times higher than the probability when using RecSplit's brute force approach.

To efficiently store $r$, we only try hash function identifiers which are multiples of $m$. This number plus $r$ is stored for each leaf. We can restore $r$ later by calculating modulo $m$ and restore the hash function index by rounding down to the next multiple of $m$. At query time, a rotation corresponds to an addition modulo $m$ to each object in the set $B$. The space overhead per object introduced by rotation fitting tends to 0 for large $m$ [5].

**Lookup Tables.** It is possible to avoid trying out all $m$ rotations by using a lookup table $t$. For all possible values of $a$, this table contains a rotation parameter $t[a]$ such that $\mathrm{rot}_m^{t[a]}(a)$ is minimal. If a value $x$ can be rotated to get the value $y$, then $\mathrm{rot}_m^{t[x]}(x) = \mathrm{rot}_m^{t[y]}(y)$. Let

$c = 2^m - 1$ be the word where the $m$ least significant bits are set. The value $\hat{b} = b \oplus c$ is $b$ with the $m$ least significant bits flipped. Note that $b$ can fill the holes in $a$ if and only if $\hat{b}$ can be rotated to match $a$. Thus, the necessary rotation of $b$ can be calculated as $r = (t[\hat{b}] - t[a]) \bmod m$ using two table lookups. Rotation $r$ is valid if $a|\text{rot}_m^r(b) = c$.

Because rotation is a very cheap operation, preliminary experiments show no improvement by lookup tables. Especially on GPUs, shared memory is a scarce resource and global memory is too slow. Our implementation therefore does not use lookup tables. Nonetheless, rotation fitting with lookup tables provides an asymptotic improvement of the running time by a factor of $m$. We also find the idea to normalize random permutations like this an interesting and novel concept. Applying this idea to other permutations is left for future research.

# 5 Parallelization

We describe the SIMD implementation in Section 5.1 and, on top of it, the multi-threaded implementation in Section 5.2. Finally, we describe the GPU implementation in Section 5.3.

## 5.1 SIMD

For the SIMD parallelization, we focus on the description of bijections and splittings, which (in most configurations) take most time of the construction. While we additionally accelerate the construction of the Elias-Fano data structure, the ideas are more straight forward and are omitted due to space constraints. The main idea of our SIMD parallelization is to try multiple hash function seeds simultaneously. Depending on the operation, we use SIMD lanes with a width of either 32 bits or 64 bits.

**Bijections.** For the bijections, each SIMD lane is responsible for trying one hash function. For this, we load consecutive hash function identifiers and the same input object to each lane of a SIMD vector, and evaluate the hash function. The resulting hash value in each lane is converted to a single bit by taking two to the power of it. After calculating the logical `OR` of these bits for all objects in the set, we check for a bijection by comparing each lane with a constant that has all $m$ lower bits set to 1. For rotation fitting, remember that the number we store as a seed is the hash function identification plus the rotation. This number should be as small as possible to avoid wasting space, so caution must be taken when trying out the rotations. If one lane finds a bijection, it might be possible that a higher rotation leads to a bijection on a lane with a smaller hash function index. Because this gives a smaller overall number to store, we always try all rotation values, even if a bijection is found.

**Splittings.** For the splittings, the original implementation uses small arrays of counters. Each counter contains the number of objects hashed to the respective split section. Instead, we use two different methods. For the *upper* aggregation levels with fanout 2, we use a single counter for the number of objects hashed to the left child. The number of objects in the right child can then be determined by subtraction. For all practical leaf sizes ($\ell \leq 24$), each counter of a valid *lower* level splitting fits into a single byte. Because an overflowing counter for one child would then just add 1 to the next counter, such overflows cannot make an invalid splitting look valid. When a seed for a valid splitting is found, we need to redistribute the objects. We now use SIMD to apply the same hash function to several objects at once, and store the results in an array. We then redistribute the objects without SIMD parallelism.

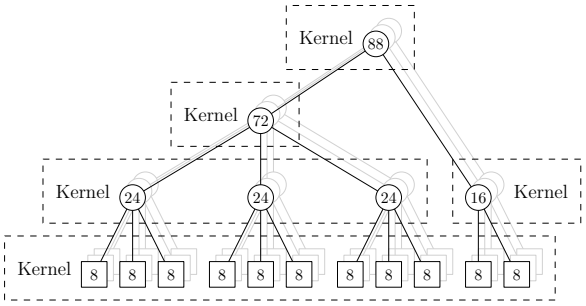



**Figure 2** Illustration of how all equally-shaped splitting trees are handled together on the GPU.

## 5.2 Multi-Threading

The original RecSplit implementation only uses a single thread. This leaves a lot of processing power unused since most modern processors contain multiple processing cores. As stated in the original RecSplit paper [14], parallelizing RecSplit is fairly easy because the buckets are completely independent of each other. First, we sort the input objects by their bucket index in parallel, and then determine the bucket borders. We then start several threads and assign a consecutive portion of the buckets to each thread. Because the number of buckets is large and the input objects are hashed to buckets uniformly, the load of all threads is reasonably balanced.

After a splitting or bijection is found, it must be stored in the Golomb-Rice coded sequence. To avoid synchronization, each thread uses its own local sequence and treats its input as if it was the complete input. This means it also stores the pointers to the start of each bucket encoding locally. After all threads are done, we sequentially concatenate the Golomb-Rice sequences and build the combined Elias-Fano data structure holding the prefix sum of bucket sizes and pointers to the bucket encodings.

## 5.3 GPU

In the GPU implementation, we first partition the objects to their buckets and partition the buckets by their respective size. We then use the GPU to determine the splittings and bijections within the buckets. Buckets with the same size have splitting trees with the same shape and can therefore be handled efficiently within the same set of kernel calls. This keeps the number of kernel calls small and is important for scalability. Using CUDA's streams, we additionally construct different bucket shapes concurrently, to utilize the GPU in case the number of buckets having a specific shape is small. For an overview, see Figure 2.

**Bijections.** All leaf nodes[1] of all trees with the same shape are constructed with a single kernel call. For each leaf node, we start one block of threads. First, the threads in each block cooperate to load all objects relevant for that leaf node into the shared memory. Similar to the SIMD implementation, where each lane tried a different hash function, now each thread tries a different hash function. After each hash function, the threads synchronize, check if a bijection was found and if it was, store the hash function index into global memory.

[1] All leaf nodes except possibly the last of each tree, which might have fewer objects.

**Splittings.** Like for the bijections, each splitting is handled by a thread block. The threads cooperate to load the objects into the shared memory and then each thread tries a different hash function index. For the two lowest aggregation levels, the thread blocks of all nodes in that level are started together using one kernel call (see Figure 2). Note that on these levels, the size of a node and the starting seed is constant. Therefore, the levels are very homogeneous. Conversely, the higher levels with fanout $s = 2$ are more heterogeneous. In particular, the number of objects on a specific level may be different for different nodes on the same level. Therefore, we launch individual kernels for each of those splittings, which contain the thread block for all trees with the same shape. We use multiplication and shifts to increment the counters of how many objects ended up in each lane. An alternative variant that stores counters in shared memory is slower in preliminary experiments, even when padding the counters to reduce the probability of bank conflicts. After a valid splitting is found, the threads in a block cooperate to reorder the objects in that node accordingly.

**Assembly.** Because the kernels are launched per level, the results are stored in BFS order. For the final data structure, we need to store them in preorder. The CPU unpacks the resulting seeds recursively and writes them to an encoded sequence.

## 6 Experiments

We first describe the experimental setup and general improvements. We then continue with a comparison of different techniques of our implementation before comparing the implementation with competitors from the literature. The code and scripts needed to reproduce our experiments are available on GitHub under the General Public License [22, 23].

**Experimental Setup.** We ran most of our experiments on an Intel i7 11700 processor with 8 cores (16 hardware threads (HT)) and a base clock speed of 2.5 GHz, supporting AVX-512. The machine runs Ubuntu 22.04 with Linux 5.15.0 and contains an NVIDIA RTX 3090 GPU. For additional experiments, we used a machine with an AMD EPYC 7702P processor with 64 cores (128 hardware threads) and a base clock speed of 2.0 GHz. The machine runs Ubuntu 20.04 with Linux 5.4.0 and supports only AVX2. Unless otherwise noted, all experiments were run on the Intel machine. We used the GNU C++ compiler version 11.2.0 with optimization flags `-O3 -march=native`. The SIMD implementation only supports x86 CPUs and is optimized towards AVX-512 using the Vector Class Library [18]. The GPU implementation uses CUDA 11. As a reminder, only the *construction* is using SIMD, multi-threading, and/or the GPU. The query implementation is identical for the SIMD and GPU implementation and almost equal to the original implementation [14]. We therefore did not compare the query performance of SIMD and GPU implementation.

For the comparison with competitors, we used strings of uniform random length $\in [10, 50]$ containing random characters except for the zero byte. Note that, as a first step, all competitors generate a *master hash code* (MHC) of each object using a high quality hash function. This makes the remaining computation largely independent of the input distribution. When only comparing different configurations of our own data structure, we used random 128-bit integers directly as MHC, which follows the approach of the original implementation [14].

### 6.1 Our Implementation

While the original implementation [14] uses std::sort to partition objects into buckets, we use IPS$^2$Ra [1]. For the less space efficient configurations ($\ell < 5, b < 100$), constructing the buckets is fast, so significant time is spent on sorting objects to buckets. For these

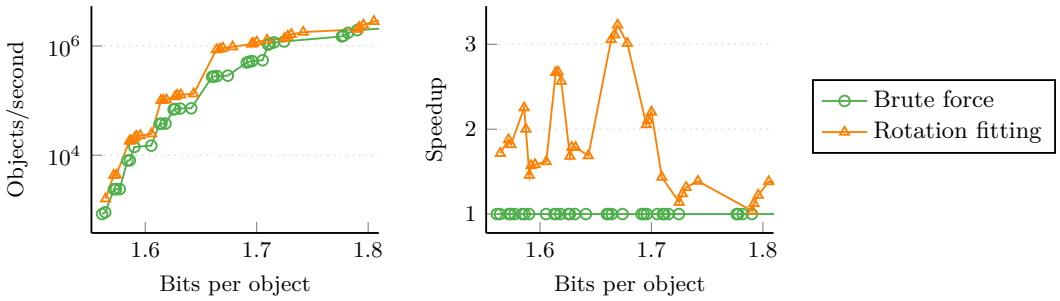



**Figure 3** Pareto front over the construction throughput of different variants of searching for bijections in the leaves. Single-threaded, non-vectorized measurements with $n = 5$ Million objects. The plot on the right gives speedups relative to the brute force method.[3]

configurations, IPS$^2$Ra both speeds up the sequential case and also enables sorting in parallel. For more space-efficient configurations ($\ell > 8$), the partitioning step needs less than 1% of the total construction time, both in the parallel and the sequential case. In this section, we compare against a slight adaption of the original implementation, using IPS$^2$Ra and supporting parallel construction.

**Rotation Fitting.** In order to compare rotation fitting with the brute force variant, we give a Pareto front[2] of space usage versus construction time in Figure 3. The construction time refers to the entire MPHF construction, including the time used for splittings. Rotation fitting is consistently faster, making the entire MPHF construction up to 3 times faster. The space overhead of rotation fitting becomes negligible for moderately large $\ell$ (see extended version [5]). Unless otherwise noted, all following experiments use rotation fitting.

**Dependence on Input Parameters.** In Figure 4, we plot the throughput of the SIMD, GPU and non-vectorized versions for different leaf sizes $\ell$ and bucket sizes $b$. For better comparability with the original paper [14], we include a wide range of configurations, even ones that are not very competitive. The SIMD version is consistently up to 4.5 times faster than the non-vectorized version and shows the same scaling behavior in $\ell$. The plot indicates that there is no configuration where one would prefer the non-vectorized version. While the GPU offers significant speedups for space efficient configurations, it performs not as good for the space inefficient configurations. A reason for this is data transfers to and from the GPU.

**Multi-Threading.** Table 1 shows that the parallel construction is up to 5 times faster on an 8-core machine. In the extended version [5], we give more detailed measurements of how different RecSplit configurations scale in the number of CPU threads. Rather unusual configurations with extremely small buckets ($b = 5$) do not scale as well, because they spend a lot of time partitioning the objects to buckets – even though we already use the highly optimized parallel sorter IPS$^2$Ra [1].

[2] A configuration is on the Pareto front if it is not dominated by any other configuration with respect to both construction time and space consumption.

[3] Note that giving speedups is non-trivial here because there might not be a configuration that achieves the same space usage that we could compare with. We therefore calculate the speedup relative to an interpolation of the next larger and next smaller data points. This is reasonable since RecSplit instances can be interpolated as well by hashing a certain fraction of objects into data structures with different configurations.

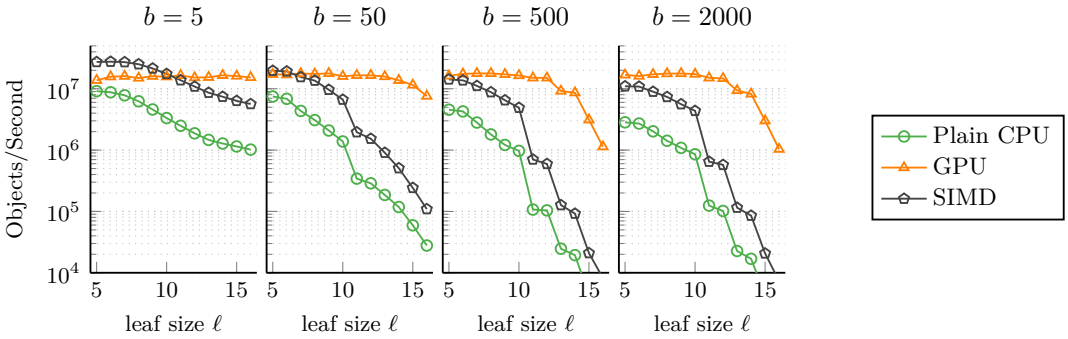



**Figure 4** Construction throughput with different hardware architectures based on different input parameters. $n = 5$ Million objects, 1 CPU thread.

**Overall Speedup.** Our rotation fitting technique leads to a speedup of up to 3 (see Figure 3) and SIMD parallelism improves the construction speed by up to a factor of 4.5 (see Figure 4). Multi-threading for highly space-efficient configurations shows a speedup of close to 5. Table 1 shows the overall improvement of our implementation on CPU and GPU when compared to the original RecSplit implementation. The original RecSplit paper says that MPHF construction at 1.56 bits per object is possible. This configuration with 5 Million objects takes about 1.5 hours using the original implementation. Our SIMD implementation achieves the same space usage in just 2 minutes on the CPU and 5 seconds on the GPU. Investing about 40 minutes of GPU time, our implementation achieves a space usage of only 1.495 bits per object. This is about 40% closer to the lower bound [2] of 1.44 bits, and simultaneously more than twice as fast as the original implementation.

**Energy Consumption.** Of course, directly comparing CPU and GPU implementations is unfair. A sensible metric to compare them is the energy consumption, which can be a major cost factor. Additionally, the energy consumption is not influenced by market prices. Table 2 gives energy consumption measurements for different configurations and hardware architectures. The energy consumption is homogeneous throughout most of the execution time, except for a short ramp-up in the beginning. We do not count the ramp-up to the energy consumption. Measurements are performed using a Voltcraft 870 Multimeter.

Even though SIMD instructions need slightly more power, the total energy consumption of constructing one MPHF is lower. The GPU, even though it needs significantly more power, is so much faster that the resulting energy usage is about 1000 times lower than the original single-threaded CPU implementation. For basic RecSplit, the AMD machine needs about 1.5 times more time than the Intel machine. This can be readily explained by a lower clock frequency. This performance gap grows to a factor 4.6 for sequential SIMDRecSplit. The likely main reason is that the AMD machines lacks the AVX-512 vector units of the Intel machine. Still, since both processors have two 256-bit AVX2 units per core, it seems that better performance might be achievable with careful tuning for the AMD architecture. On the contrary, the AMD machine shows good scalability so that the energy consumption when using the entire machine is only a factor 1.3 larger than on the Intel machine – despite the fact that our implementation was tuned for the Intel architecture.

**Table 1** Construction time of the GPU implementation compared to our multi-threaded adaption of the original RecSplit implementation. $n = 5$ Million objects (strong scaling). Construction times are given in $\mu$s/object. We do not report speedups for $\ell = 24$ because the CPU baseline takes too long for this configuration.

| Configuration | Method | Bijections | Threads | B/Obj | Constr. | Speedup |
|---|---|---|---|---|---|---|
| $\ell = 16, b = 2000$ | RecSplit [14] | Brute force | 1 | 1.560 | 1175.4 | 1 |
| | RecSplit | Brute force | 16 | 1.560 | 206.5 | 5 |
| | SIMDRecSplit | Rotation fitting | 1 | 1.560 | 138.0 | 8 |
| | SIMDRecSplit | Rotation fitting | 16 | 1.560 | 27.9 | 42 |
| | GPURecSplit | Brute force | GPU | 1.560 | 1.8 | 655 |
| | GPURecSplit | Rotation fitting | GPU | 1.560 | 1.0 | 1173 |
| $\ell = 18, b = 50$ | RecSplit [14] | Brute force | 1 | 1.707 | 2942.9 | 1 |
| | RecSplit | Brute force | 16 | 1.713 | 504.0 | 5 |
| | SIMDRecSplit | Rotation fitting | 1 | 1.709 | 58.3 | 50 |
| | SIMDRecSplit | Rotation fitting | 16 | 1.708 | 12.3 | 239 |
| | GPURecSplit | Brute force | GPU | 1.708 | 5.2 | 564 |
| | GPURecSplit | Rotation fitting | GPU | 1.709 | 0.5 | 5438 |
| $\ell = 24, b = 2000$ | GPURecSplit | Brute force | GPU | 1.496 | 2300.9 | — |
| | GPURecSplit | Rotation fitting | GPU | 1.496 | 467.9 | — |

## 6.2 Comparison with Competitors

We now compare our implementation to the sequential codes RecSplit [14], SicHash [30], and CHD [2] as well as the parallel codes PTHash [37], PTHash-HEM [36] and BBHash [31].

**Space usage trade-off.** Figure 5 shows a space versus construction time Pareto front for each approach. Looking at a single thread first, we make the surprising observation that SIMDRecSplit not only wins for the most space efficient configurations for which we designed it but, by far, dominates all the other methods also for less space-efficient cases. For parallel construction, SIMDRecSplit even strengthens its margin to the competing approaches.

**Construction Scaling.** Figure 6 compares scaling behavior of the parallel codes. We see that BBHash scales poorly while both PTHash-HEM and SIMDRecSplit scale well on the 8-core Intel machine. However, SIMDRecSplit scales better than PTHash-HEM on the 64-core AMD machine.

**Queries.** Table 3 shows that when looking at the query time, PTHash is a clear winner. While BBHash can achieve the same query speed and good construction speed, its space usage is large. SicHash has a query time close to PTHash's most compact representation, but is faster to construct and more space efficient. All RecSplit variants can achieve significantly lower space than other competitors but require considerably more query time. Our single-threaded SIMD implementation dominates most competitors with respect to both space and construction time. The use of rotations makes the queries about 10% slower than the original RecSplit implementation. The main goal of RecSplit is to achieve extremely small representation, and queries are not very fast to begin with, so this seems acceptable.

**Table 2** Energy consumption with $\ell = 18, b = 50$ and $n = 5$ Million objects. Energy consumption is both given as difference to the idle power, as well as total energy consumption of the whole system. For CPU-only measurements of the 8-core Intel machine, we dismount the GPU.

| | | | | Total system | | Δ to idle | |
|---|---|---|---|---|---|---|---|
| Machine | Method | Threads | Constr. Seconds | Power Watt | Energy Joule | Power Watt | Energy Joule |
| 8-core Intel | RecSplit [14] | 1 | 14 714.5 | 78 | 1 147 731 | 37 | 544 436 |
| | SIMDRecSplit | 1 | 291.5 | 87 | 25 360 | 46 | 13 409 |
| | SIMDRecSplit | 16 | 61.5 | 104 | 6 396 | 63 | 3 874 |
| | GPURecSplit | | 2.5 | 457 | 1 142 | 380 | 950 |
| 64-core AMD | RecSplit [14] | 1 | 21 620.8 | 223 | 4 821 438 | 91 | 1 967 492 |
| | SIMDRecSplit | 1 | 1 328.7 | 224 | 297 629 | 92 | 122 240 |
| | SIMDRecSplit | 128 | 23.6 | 364 | 8 590 | 232 | 5 475 |

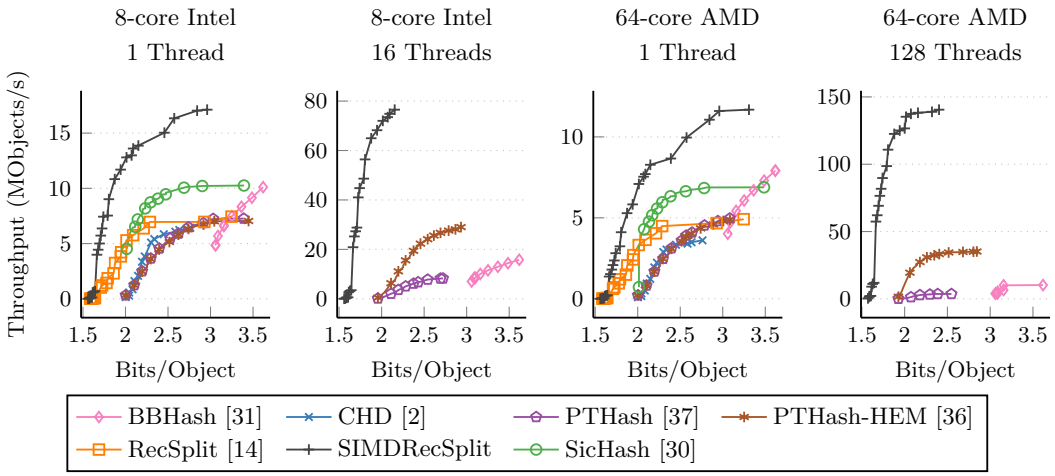



**Figure 5** Trade-off of construction time vs space usage. Weak scaling, $n/p = 10$ Million objects. For SicHash and PTHash, we plot all Pareto optimal data points but only show markers for every fourth point to increase readability. Therefore, the lines might bend on positions without markers.

## 7 Conclusion and Future Work

We have shown that by harnessing parallelism at all available levels – bits, vectors, cores, and GPUs – one can dramatically accelerate the construction of highly space efficient minimal perfect hash functions (MPHFs) using the brute force RecSplit approach [14]. This leads to speedups of up to 239 on SIMD and 5438 on the GPU and also dramatically reduces energy consumption. Surprisingly, this even turns out to be the fastest available approach for constructing less space-efficient MPHFs. This is not what we expected. Our initial hypothesis was that there would be a trade-off with asymptotically faster approaches winning for fewer requirements on space consumption. Our new technique *rotation fitting* reduces the work needed per tried hash function while adding a tiny bit of space requirement. The asymptotically "obvious" improvement of replacing $\ell$ rotations/checks by two table lookups are not productive on current architectures. So, brute force, simplicity (in the inner loops), and parallelism currently wins against any attempt at algorithmic sophistication.

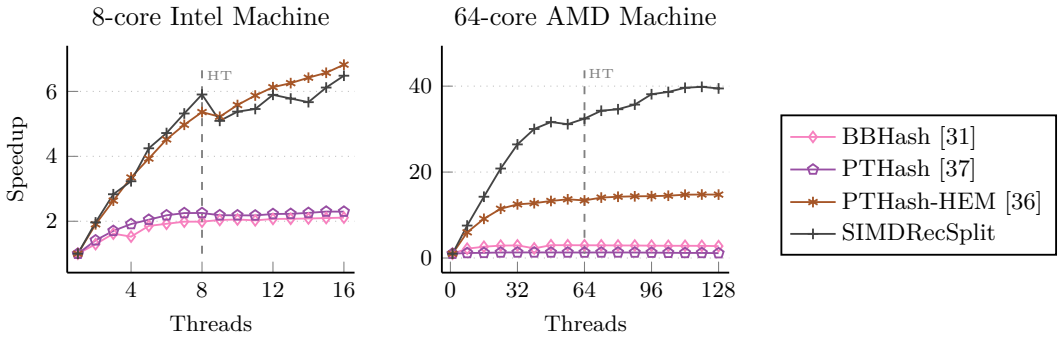



**Figure 6** Construction time speedups when using multiple threads $t$. Strong scaling, $n = 50$ Million. Speedups are given relative to each method's single threaded performance. For a comparison of absolute performance, refer to Figure 5. Configurations used are BBHash: $\gamma = 2.0$; PTHash/PTHash-HEM: $c = 6.0$, $\alpha = 0.95$, EF; SIMDRecSplit: $\ell = 10$, $b = 2000$.

**Table 3** Query and construction time of different competitor configurations on 10 Million objects.

| Method | Bits/Obj. | Constr./Obj. | Query/Obj. |
|---|---|---|---|
| BBHash [31], $\gamma$=5.0 | 6.871 | 50 ns | 36 ns |
| BBHash [31], $\gamma$=1.0 | 3.059 | 208 ns | 51 ns |
| PTHash [37], $c$=11.0, $\alpha$=0.88, D-D | 4.379 | 138 ns | 25 ns |
| PTHash [37], $c$=7.0, $\alpha$=0.99, C-C | 3.313 | 199 ns | 20 ns |
| PTHash [37], $c$=6.0, $\alpha$=0.99, EF | 2.345 | 248 ns | 35 ns |
| SicHash [30], $\alpha$=0.9, $p_1$=20, $p_2$=77 | 2.412 | 119 ns | 41 ns |
| SicHash [30], $\alpha$=0.97, $p_1$=44, $p_2$=30 | 2.081 | 172 ns | 40 ns |
| RecSplit [14], $\ell$=5, $b$=5 | 2.928 | 145 ns | 65 ns |
| RecSplit [14], $\ell$=8, $b$=100 | 1.793 | 709 ns | 75 ns |
| RecSplit [14], $\ell$=14, $b$=2000 | 1.584 | 126 534 ns | 96 ns |
| SIMDRecSplit, $\ell$=5, $b$=5 | 2.96 | 49 ns | 71 ns |
| SIMDRecSplit, $\ell$=8, $b$=100 | 1.806 | 107 ns | 80 ns |
| SIMDRecSplit, $\ell$=14, $b$=2000 | 1.585 | 11 742 ns | 110 ns |

Another attempt at sophistication that so far failed is to combine brute force RecSplit with the retrieval approach of SicHash [30]. The idea of this *ShockHash* approach is to allow retrieval of a single bit of information for each element. The brute force part then tries pairs of random hash functions until they define a pseudo-forest – a collection of components consisting of a tree plus one additional edge. While ShockHash seems to allow space efficient perfect hashing, initial experiments indicated that performance-wise ShockHash is also inferior to pure brute force (see the extended version [5] for details). More efficient implementations of ShockHash may change this picture in the future.

Also, rotation fitting could be generalized by splitting into more than two parts. The resulting search for several rotations gives more room for sophistications like search space pruning. Furthermore, the approach from rotation fitting to use a lookup table for normalizing bit patterns could be generalized to a richer set of mappings than just rotations.

Everything discussed so far is mainly concerned with construction time. However, an equally important problem is to improve query time. Traversing an aggressively compressed tree for each query is inherently more expensive than the simple constant time operations

needed in PTHash [37] or SicHash [30] but there should be more efficient ways to break down MPHF construction into small subproblems that can be solved with brute force. We believe that the techniques developed here will turn out to be useful in that respect.

Finally, we can look for generalizations of RecSplit for computing non-minimal PHFs which allows us to further reduce space consumption of the hash function itself. Better tuning of the SIMD variant for AMD or perhaps even a portable implementation that also works on ARM or RISC-V would be relevant for widespread application.

## References


1 Michael Axtmann, Sascha Witt, Daniel Ferizovic, and Peter Sanders. Engineering in-place (shared-memory) sorting algorithms. *ACM Trans. Parallel Comput.*, 9(1):2:1–2:62, 2022. doi:10.1145/3505286.

2 Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2009. doi:10.1007/978-3-642-04128-0_61.

3 Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. In *FOCS*, pages 182–193. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00026.

4 Dominik Bez. Perfect hash function generation on the GPU with RecSplit. Master's thesis, Karlsruhe Institute for Technology (KIT), 2022. doi:10.5445/IR/1000152719.

5 Dominik Bez, Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. High performance construction of recsplit based minimal perfect hash functions, 2022. doi:arXiv:2212.09562.

6 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Perfect hashing for data management applications. *CoRR*, abs/cs/0702159, 2007. arXiv:0702159.

7 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Simple and space-efficient minimal perfect hash functions. In *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2007. doi:10.1007/978-3-540-73951-7_13.

8 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, 38(1):108–131, 2013. doi:10.1016/J.IS.2012.06.002.

9 Jarrod A. Chapman, Isaac Ho, Sirisha Sunkara, Shujun Luo, Gary P. Schroth, and Daniel S. Rokhsar. Meraculous: De novo genome assembly with short paired-end reads. *PLOS ONE*, 6(8):1–13, August 2011. doi:10.1371/journal.pone.0023501.

10 David Richard Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, Canada, 1996.

11 Zbigniew J. Czech, George Havas, and Bohdan S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Inf. Process. Lett.*, 43(5):257–264, 1992. doi:10.1016/0020-0190(92)90220-P.

12 Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. In *SEA*, volume 233 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SEA.2022.4.

13 Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974. doi:10.1145/321812.321820.

14 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. Recsplit: Minimal perfect hashing via recursive splitting. In *ALENEX*, pages 175–185. SIAM, 2020. doi:10.1137/1.9781611976007.14.

15 Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo filter: Practically better than bloom. In *CoNEXT*, pages 75–88. ACM, 2014. doi:10.1145/2674005.2674994.

16 Robert Mario Fano. On the number of bits required to implement an associative memory. Technical report, MIT, Computer Structures Group, 1971. Project MAC, Memorandum 61.

17 Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Computers*, 21(9):948–960, 1972. doi:10.1109/TC.1972.5009071.

18 Agner Fog. C++ vector class library. http://www.agner.org/optimize/vectorclass.pdf, 2013.

19 Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005. doi:10.1007/S00224-004-1195-X.

20 Edward A. Fox, Qi Fan Chen, and Lenwood S. Heath. A faster algorithm for constructing minimal perfect hash functions. In *SIGIR*, pages 266–273. ACM, 1992. doi:10.1145/133160.133209.

21 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

22 GpuRecSplit – GitHub. https://github.com/ByteHamster/GpuRecSplit, 2023.

23 MPHF-Experiments – GitHub. https://github.com/ByteHamster/MPHF-Experiments, 2023.

24 Solomon W. Golomb. Run-length encodings (corresp.). *IEEE Trans. Inf. Theory*, 12(3):399–401, 1966. doi:10.1109/TIT.1966.1053907.

25 Intel. Advanced vector extensions programming reference. https://www.intel.com/content/dam/develop/external/us/en/documents/36945, 2011.

26 Intel. Avx-512 instructions. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-avx-512-instructions.html, 2013.

27 Guy Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63533.

28 Florian Kurpicz. Engineering compact data structures for rank and select queries on bit vectors. In *SPIRE*, volume 13617 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2022. doi:10.1007/978-3-031-20643-6_19.

29 Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. *ACM Trans. Graph.*, 25(3):579–588, 2006. doi:10.1145/1141911.1141926.

30 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. SicHash – small irregular cuckoo tables for perfect hashing. In *ALENEX*, pages 176–189. SIAM, 2023. doi:10.1137/1.9781611977561.CH15.

31 Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. In *SEA*, volume 75 of *LIPIcs*, pages 25:1–25:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.SEA.2017.25.

32 Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and perfect hashing using fingerprinting. In *SEA*, volume 8504 of *Lecture Notes in Computer Science*, pages 138–149. Springer, 2014. doi:10.1007/978-3-319-07959-2_12.

33 Nvidia. Nvidia ampere GA102 GPU architecture. https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf, 2020.

34 Nvidia. CUDA C++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2022.

35 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.

36 Giulio Ermanno Pibiri and Roberto Trani. Parallel and external-memory construction of minimal perfect hash functions with pthash. *CoRR*, abs/2106.02350, 2021. arXiv:2106.02350.

37 Giulio Ermanno Pibiri and Roberto Trani. PTHash: Revisiting FCH minimal perfect hashing. In *SIGIR*, pages 1339–1348. ACM, 2021. doi:10.1145/3404835.3462849.

38 Robert F. Rice. Some practical universal noiseless coding techniques. *Jet Propulsion Laboratory, JPL Publication*, 1979.

39 Sebastiano Vigna. Broadword implementation of rank/select queries. In *WEA*, volume 5038 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008. doi:10.1007/978-3-540-68552-4_12.

40 Sean A. Weaver and Marijn Heule. Constructing minimal perfect hash functions using SAT technology. In *AAAI*, pages 1668–1675. AAAI Press, 2020.

# On the Giant Component of Geometric Inhomogeneous Random Graphs

**Thomas Bläsius**
Karlsruhe Institute of Technology, Germany

**Tobias Friedrich**
Hasso Plattner Institute, University of Potsdam, Germany

**Maximilian Katzmann**
Karlsruhe Institute of Technology, Germany

**Janosch Ruff**
Hasso Plattner Institute, University of Potsdam, Germany

**Ziena Zeif**
Hasso Plattner Institute, University of Potsdam, Germany

## Abstract

In this paper we study the threshold model of *geometric inhomogeneous random graphs* (GIRGs); a generative random graph model that is closely related to *hyperbolic random graphs* (HRGs). These models have been observed to capture complex real-world networks well with respect to the structural and algorithmic properties. Following comprehensive studies regarding their *connectivity*, i.e., which parts of the graphs are connected, we have a good understanding under which circumstances a *giant* component (containing a constant fraction of the graph) emerges.

While previous results are rather technical and challenging to work with, the goal of this paper is to provide more accessible proofs. At the same time we significantly improve the previously known probabilistic guarantees, showing that GIRGs contain a giant component with probability $1 - \exp(-\Omega(n^{(3-\tau)/2}))$ for graph size $n$ and a degree distribution with power-law exponent $\tau \in (2, 3)$. Based on that we additionally derive insights about the connectivity of certain induced subgraphs of GIRGs.

**2012 ACM Subject Classification** Theory of computation → Random network models

**Keywords and phrases** geometric inhomogeneous random graphs, connectivity, giant component

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.20

## 1 Introduction

Geometric inhomogeneous random graphs (GIRGs) are a generative graph model where vertices are weighted and placed in a geometric ground space and the probability for two of them to be adjacent depends on the product of their weights, as well as their distance [19]. In a sense the model combines the strengths of *inhomogeneous random graphs* [28] and *random geometric graphs* [26]. Introduced as a simplified and more general version of *hyperbolic random graphs* (HRGs) [23], GIRGs share crucial properties with complex real-world networks. Such networks are typically characterized by a *heterogeneous degree distribution* (with few high-degree vertices, while the majority of vertices has small degree), *high clustering* (vertices with common neighbors are likely adjacent themselves), and a *small diameter* (longest shortest path), and it has been shown that GIRGs and HRGs capture these properties well [17, 19, 25].

Beyond these structural properties, GIRGs have also been observed to be a good model for real-wold networks when it comes to the performance of graph algorithms [3]. This makes the GIRG framework relevant for algorithmic purposes in multiple ways. On the one hand, they are a useful tool in the context of average-case analysis, where they yield more


realistic instances than, e.g., the Erdős–Rényi model, while it is still sufficiently simple to be mathematically accessible [3, 7]. On the other hand, we can use GIRGs to generate an abundance of benchmark instances with varying properties, allowing us to perform thorough evaluations of algorithms even when real-world data is scarce [4, 5].

One of the most basic graph properties, which is also relevant from an algorithmic point of view, is *connectivity*, i.e., the question about what parts of a graph are connected via paths. For random graphs, the first question that typically arises in the context of connectivity revolves around the emergence of a so-called *giant component*, which is a connected component whose size is linear in the size of the graph. The existence of a giant has been researched on many related graph models like *Erdős–Rényi random graphs* [12, 13], *random geometric graphs* [2, 11, 26, 18], as well as on *Chung-Lu random graphs* that also capture inhomogeneous random graphs [1, 9, 10].

Unsurprisingly, being such a fundamental feature, connectivity has also been studied on GIRGs, and since HRGs are so closely related to them, we consider the corresponding results to be relevant here as well. For HRGs we know how the emergence of a giant depends on certain model properties that control the degrees of the resulting graph [6, 14]. We note that some analyses there are based on a coupling from HRGs to a continuum percolation model that exhibits a strong resemblance to GIRGs (see [14, Section 2] and [19, Part I, Section 3.5]). Beyond the giant we also have bounds on the size of the second largest component of HRGs [20]. For GIRGs it is known that a giant exists *asymptotically almost surely*, i.e., with probability $1 - o(1)$ [21], with another proof giving a certainty of $1 - n^{-\omega(1)}$ where $n$ denotes the number of vertices in the graph [19, Theorem 4.2]. We note that the specific function in the exponent has not been determined before.

In this paper, we answer this question, by showing that threshold GIRGs have a giant component with probability at least $1 - \exp(-\Omega(n^{(3-\tau)/2}))$. This improves the previous results in two ways. First, our proof is simpler and shorter than the technical existing proofs for HRGs [6, 14]. Secondly, our probability bound is substantially stronger compared to previous bounds obtained for GIRGs. Moreover, we note that our improved bound does not only hold for the full graph but also translates to subgraphs located in restricted regions of the ground space. The argument for this is inspired by a technique used for HRGs [14, Section 4] (though it is much simpler in our case).

Besides providing more accessible insights in the connectivity of GIRGs, we believe that our results, in particular those on subgraphs in restricted regions, can be helpful for algorithmic applications. For example in problems like *balanced connected partitioning* [8], one is interested in partitioning a graph into connected components of (roughly) equal size and in *component order connectivity* [16] the goal is to find a small separator that divides the graph into components of bounded size. There it is important, that the graph cannot only be separated into smaller pieces but that these pieces remain actually connected.

In the following, we give a brief overview of the basic concepts used in the paper (Section 2) before presenting our proofs regarding the emergence of a giant in GIRGs (Section 3).

## 2 Preliminaries

**Geometric Inhomogeneous Random Graphs.** Let $\mathbb{B}^d = [0,1]^d$ be the $d$-dimensional hypercube ($\mathbb{B}$ for "box") and let dist be the $L_\infty$ metric, i.e., for $x = (x_1, \dots, x_d) \in \mathbb{B}^d$ and $y = (y_1, \dots, y_d) \in \mathbb{B}^d$ we have $\text{dist}(x, y) = \max_{i \in [d]} |x_i - y_j|$.

A *geometric inhomogeneous random graph (GIRG)* $G = (V, E)$ with *ground space* $\mathbb{B}^d$ is obtained in three steps. The first step consists of a homogeneous Poisson point process on $\mathbb{B}^d$, with an intensity that yields $n$ points in expectation. Each point is then considered to

be a vertex in the graph. In the second step, each vertex $v$ is assigned a *weight* $w_v > 1$ that is sampled according to a Pareto distribution with exponent $\tau \in (2,3)$, i.e., $\Pr[w_v \leq w] = 1 - w^{-(\tau-1)}$. In the third step, any two vertices $u$ and $v$ are connected by an edge with a probability that depends on their distance and their weights. More precisely, there are two variants. In a *threshold GIRG*, $u$ and $v$ are adjacent if and only if

$$\operatorname{dist}(u,v) \leq \left(\frac{\lambda w_u w_v}{n}\right)^{1/d},$$

where the constant $\lambda > 0$ controls the expected average degree of the graph. We note that the relation between $\lambda$ and the corresponding average degree is not trivial and refer to [5, Section 4.3] for details. In the *temperate* variant we have an additional temperature parameter $T \in (0,1)$ and the probability for $u$ and $v$ to be adjacent is given by

$$\Pr[\{u,v\} \in E] = \min\left\{1, \left(\frac{\lambda w_u w_v}{n \cdot (\operatorname{dist}(u,v))^d}\right)^{1/T}\right\}.$$

The threshold variant is the limit of the temperature variant for $T \to 0$. We denote the resulting probability distribution of graphs with $\mathcal{G}(n, \mathbb{B}^d, \tau, \lambda, T)$ for general GIRGs (allowing temperatures in $T \in [0,1)$). When we just refer to the threshold case, we use $\mathcal{G}(n, \mathbb{B}^d, \tau, \lambda)$. We assume the parameters $d, \tau, \lambda$, and $T$ to be constant, i.e., independent of $n$.

**GIRG Variants.** In the literature, several variants of the GIRG model have been studied and we want to briefly discuss the choice we made here. Usually, GIRGs are considered with a torus $\mathbb{T}^d$ as ground space, i.e., the distance in the $i$th dimension, between $x$ and $y$ is $\min\{|x_i - y_i|, 1 - |x_i - y_i|\}$ instead of just $|x_i - y_i|$. The torus usually makes arguments easier as it eliminates the special case close to the boundary of $\mathbb{B}^d$. However, in our case, this is not relevant. Moreover, as distances in $\mathbb{T}^d$ are only smaller than in $\mathbb{B}^d$, all our results concerning the largest connected component directly translate to the case where $\mathbb{T}^d$ is the ground space.

Moreover, instead of sampling $n$ points uniformly at random in the ground space, we use a Poisson point process. This is a technique often used in geometric random graphs as it makes the number of vertices appearing in disjoint regions stochastically independent. This is a similar difference as the one between the Erdős–Rényi model $G(n,m)$ with a fixed number of edges $m$ and the Gilbert model $G(n,p)$ with a fixed probability $p$ for each individual edge to exist. While we generally advocate for using the Poisson variant of the GIRG model, we note that our result carries over to the uniform model.

**Poisson Point Process.** Let $R \subseteq \mathbb{B}^d$ be a region of the ground space with volume $a$. Then, the size of the vertex set $V(R)$, i.e., the number of vertices that are sampled in $R$ is a random variable following a Poisson distribution with expectation $\mu = an$. This in particular means that the probability for $R$ to contain no vertex is $\exp(-\mu)$.

We note that the Poisson point process we consider is a *marked* process, where each point sampled from $\mathbb{B}^d$ obtains a weight sampled from a weight space $\mathcal{W}$ as a mark. Due to the marking theorem, this is equivalent to considering an (inhomogeneous) Poisson point process of the product space $\mathbb{B}^d \times \mathcal{W}$, i.e., colloquially speaking, each point pops up with a position and a weight instead of initially only having a position and drawing the weight as an afterthought. This is also equivalent to just sampling the number of points $N$ following a Poisson distribution and viewing the positions and the weights as marks that are sampled subsequently for each of the $N$ points. Throughout the paper, we switch between these different perspectives without making this explicit.

**Lowest Weights Dominate.** We regularly consider weight ranges $[w_1, w_2]$ with $w_2 \geq c \cdot w_1$ for a constant $c > 1$. The probability for a $v$ to have weight in $[w_1, w_2]$ is dominated by $w_1$:

$$\Pr[w_v \in [w_1, w_2]] = w_1^{-(\tau-1)} - w_2^{-(\tau-1)} \geq w_1^{-(\tau-1)} \cdot \left(1 - c^{-(\tau-1)}\right) \in \Theta(\Pr[w_v \geq w_1]).$$

# 3 Existence of a Giant Component

We want to show that a threshold GIRG is highly likely to contain a connected component of linear size. Our argument goes roughly as follows. We first note that vertices with weight at least $\sqrt{n/\lambda}$ form a clique, which we call the *core* of the graph. For each non-core vertex, we can show that the probability that it has a path into the core is non-vanishing, i.e., it is lower bounded by a non-zero constant. This already shows that we get a connected graph of linear size in expectation.

To show concentration, i.e., that we get a large connected component with the claimed probability, we essentially need to show that the events for different low-weight vertices to connect to the core are sufficiently independent of each other. To this end, we subdivide the ground space into a grid of regular *cells* of side length $\Delta$. We call a cell *nice* if a linear number of its vertices connect to the core via paths not leaving the cell and then show that a cell is nice with non-vanishing probability. As this only considers paths within the cell, the different cells are independent. Thus, we get a series of independent coin flips, one for each cell. If a constant fraction of these coin flips succeeds, we have a connected component of linear size. Hence, if the number of cells is sufficiently large, we get concentration via a Chernoff bound. It follows that we essentially want to choose the cell width $\Delta$ to be as small as possible such that cells are still nice with non-vanishing probability.

In Section 3.1, we first show that every vertex has constant probability to have a path to the core. In fact, we show something slightly stronger, by considering not just any paths but so-called layer paths. Afterwards, we use this result in Section 3.2, to bound the probability for a cell to be nice. This then also informs us on how to choose the cell width $\Delta$ and thus on how many cells we obtain. With this, we can wrap up the argument in Section 3.3 by applying a Chernoff bound. Besides our main results, we there also mention immediate implications.

## 3.1 Layer Paths

We want to show that, for any individual vertex, the probability that it has a path to a vertex of the core is non-vanishing. For this, we define the $\ell$-th *layer* $V_\ell$ to be the set of vertices with weight in $[e^{\ell/2}, e^{(\ell+1)/2})$. Note that the upper and lower bounds are a constant factor apart and thus (as mentioned in Section 2) the probability for a vertex to have layer $\ell$ is asymptotically dominated by the lower bound, i.e., $\Pr[v \in V_\ell] \in \Theta(\Pr[w_v \geq e^{\ell/2}]) = \Theta(e^{-\ell(\tau-1)/2})$.

A path $(v_0, \dots, v_k)$ is a *layer path* if it goes from one layer to the next in every step, i.e., $v_i \in V_\ell$ implies $v_{i-1} \in V_{\ell-1}$ for every $i \in [k]$. Note that vertices in layer $\lceil \log(n/\lambda) \rceil$ have weight at least $\sqrt{n/\lambda}$ and thus belong to the core. Thus, the following lemma shows that each vertex has a layer path to the core with non-vanishing probability.

▶ **Lemma 1.** *Let $G \sim \mathcal{G}(n, \mathbb{B}^d, \tau, \lambda)$ be a threshold GIRG and let $v$ be a non-core vertex. The probability that there is a layer path from $v$ to layer $\lceil \log(n/\lambda) \rceil$ is non-vanishing.*

**Proof.** We bound the probability that such a layer path exists in three steps. First, we bound the probability that a vertex $u$ on layer $\ell$ has a neighbor in layer $\ell + 1$. In the second step, we consider the intersection of the events where this happens on all considered layers. Finally, we show that the resulting probability is non-vanishing.

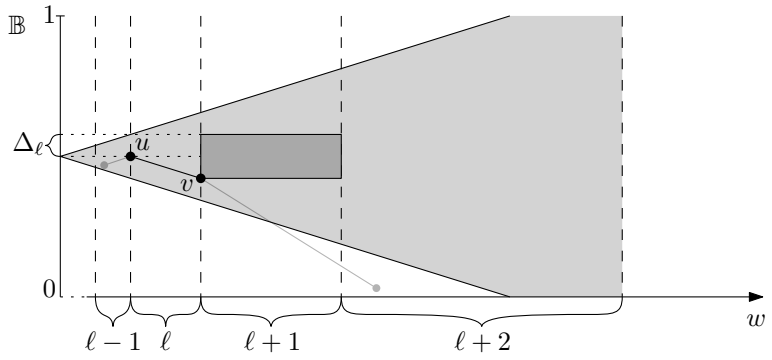



**Figure 1** Excerpt of a one-dimensional GIRG with the weights on the $x$-axis and the ground space $\mathbb{B}$ on the $y$-axis. A layer path spans from layer $\ell-1$ to $\ell+2$. The gray region is the neighborhood of vertex $u$. The dark-gray region contains all vertices in layer $\ell+1$ that have distance at most $\Delta_\ell$ to $u$.

For the first step, consider two vertices $u \in V_\ell$ and $v \in V_{\ell+1}$ in consecutive layers, as shown in Figure 1. Both their weights are at least $w = e^{\ell/2}$. Thus, they are definitely adjacent if their distance $\text{dist}(u,v)$ satisfies

$$\text{dist}(u,v) \leq \left(\frac{\lambda w^2}{n}\right)^{1/d} = \lambda^{1/d}\left(\frac{e^\ell}{n}\right)^{1/d} =: \Delta_\ell.$$

If vertex $u \in V_\ell$ is the current vertex from which we want to make the next step in a layer path, we are thus interested in the probability that there is a vertex $v$ that lies in layer $\ell+1$ with $\text{dist}(u,v) \leq \Delta_\ell$. Since these two events (being in layer $\ell+1$ and having sufficiently low distance) are independent, the probability that both happen is $\Pr[v \in V_{\ell+1}] \cdot \Pr[\text{dist}(u,v) \leq \Delta_\ell]$. As mentioned above, we have $\Pr[v \in V_{\ell+1}] \in \Theta(e^{-\ell(\tau-1)/2})$. Moreover, $\Pr[\text{dist}(u,v) \leq \Delta_\ell] \in \Theta(\Delta_\ell^d) = \Theta(e^\ell/n)$. Hence, we obtain

$$\Pr[v \in V_{\ell+1}] \cdot \Pr[\text{dist}(u,v) \leq \Delta_\ell] \in \Theta\left(e^{-\ell(\tau-1)/2} \cdot e^\ell/n\right) = \Theta\left(e^{\ell(3-\tau)/2}/n\right).$$

To conclude the first step of the proof, let $X_\ell$ be the number of vertices in layer $\ell+1$ with distance at most $\Delta_\ell$ to $u \in V_\ell$. By the above probability, we have $\mathbb{E}[X_\ell] = \Theta(e^{\ell(3-\tau)/2})$. We consider the event $X_\ell > 0$ and call it $A_\ell$. Note that $A_\ell$ implies that $u$ has at least one neighbor in the next layer. As $X_\ell$ follows a Poisson distribution, we get

$$\Pr[A_\ell] = 1 - \Pr[X_\ell = 0] = 1 - \exp(-\mathbb{E}[X_\ell]) = 1 - \exp\left(-\Theta\left(e^{\ell(3-\tau)/2}\right)\right).$$

In the second step of the proof, we now consider the intersection of all the independent events $A_0, A_1, \dots, A_{\lceil \log(n/\lambda)\rceil}$, which is sufficient for a layer path starting in layer 0 to exist. Note that a lower bound for the probability of this intersection also gives a lower bound for the existence of a layer path starting in any other layer. To show that this intersection occurs with non-vanishing probability, we utilize the fact that $\Pr[A_\ell]$ approaches 1 very quickly for increasing $\ell$. More precisely, we show that for a constant $c$, all subsequent events $A_\ell$ with $\ell \geq c$ are sufficiently likely, that we can simply take the union bound over their complements. Thus, we obtain

$$\Pr\left[\bigcap_{\ell=0}^{\lceil \log(n/\lambda)\rceil} A_\ell\right] = \Pr\left[\bigcap_{\ell=0}^{c-1} A_\ell\right] \cdot \Pr\left[\bigcap_{\ell=c}^{\lceil \log(n/\lambda)\rceil} A_\ell\right].$$

Clearly, the first factor is non-vanishing as it is the product of constantly many non-zero constants. For the second factor, we consider the complementary events and apply the union bound to obtain

$$\begin{aligned}\Pr\left[\bigcap_{\ell=c}^{\lceil\log(n/\lambda)\rceil} A_\ell\right] &= 1 - \Pr\left[\bigcup_{\ell=c}^{\lceil\log(n/\lambda)\rceil} A_\ell^C\right] \\ &\geq 1 - \sum_{\ell=c}^{\lceil\log(n/\lambda)\rceil} (1 - \Pr[A_\ell]) \\ &= 1 - \sum_{\ell=c}^{\lceil\log(n/\lambda)\rceil} \exp\left(-\Theta(e^{\ell(3-\tau)/2})\right).\end{aligned}$$

Since the sum converges, we can choose $c$ to be a sufficiently large constant such that the sum is bounded by any constant $\varepsilon > 0$. The above expression is thus at least $1 - \varepsilon$, which is non-vanishing. ◀

Observe that Lemma 1 already shows that the expected number of vertices with a layer path to the core is linear. Thus, the expected size of the connected component including the core vertices is linear. To show concentration, we separate the ground space into cells that are then considered independently.

## 3.2 A Coin Flip for Each Cell

We subdivide the ground space into a grid of regular cells of side length $\Delta$. We first show that the high-weight vertices of each cell are likely to induce a connected graph. This is useful as we can afterwards focus on vertices of lower weight. As edges between low-weight vertices are short, layer paths on these vertices can cover only a small distance and thus only few of them leave their cell, which makes different cells (mostly) independent.

▶ **Lemma 2.** *Let $G \sim \mathcal{G}(n, \mathbb{B}^d, \tau, \lambda)$ be threshold GIRG, let $C$ be a cell of side length $\Delta$, and let $w$ be a weight. Then, the graph induced by vertices in $C$ of weight at least $w$ is connected with probability at least*

$$1 - \frac{(2\Delta)^d}{\lambda w^2} \cdot \exp\left(-\frac{\lambda w^{3-\tau}}{2^d}\right) \cdot n.$$

**Proof.** We discretize the cell $C$ into sub-cells, such that vertices in adjacent sub-cells are adjacent themselves, as shown in Figure 2. Note that two vertices $u, v$ with weights $w_u, w_v \geq w$ are adjacent if their distance is bounded by

$$\operatorname{dist}(u, v) \leq \left(\frac{\lambda w^2}{n}\right)^{1/d}.$$

Thus, all vertices in adjacent sub-cells are guaranteed to be adjacent, if the side length of a sub-cell is

$$\Delta_w = \frac{1}{2}\left(\frac{\lambda w^2}{n}\right)^{1/d}.$$

Note that for very large $w$, we get $\Delta_w \geq \Delta$, in which case all vertices in $C$ are pairwise adjacent with probability 1. In the following, we therefore assume that $w$ is smaller. For a given sub-cell $C'$, we compute the probability for a given vertex $v$ to lie in $C'$ as

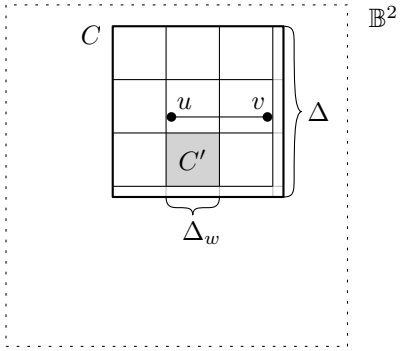



**Figure 2** The cell $C$ of width $\Delta$ is divided into sub-cells of width $\Delta_w$. The sub-cell $C'$ is completely contained in $C$. The vertices $u$ and $v$ are in adjacent sub-cells and are therefore adjacent themselves.

$$\Pr[v \in V(C')] = (\Delta_w)^d = \left(\frac{1}{2}\left(\frac{\lambda w^2}{n}\right)^{1/d}\right)^d = \frac{\lambda w^2}{2^d n}.$$

Additionally, the probability for $v$ to have weight at least $w_v \geq w$, is given by

$$\Pr[w_v \geq w] = 1 - \Pr[w_v \leq w] = w^{-(\tau-1)}.$$

Together, we obtain

$$\Pr[v \in V(C') \wedge w_v \geq w] = \Pr[v \in V(C')] \cdot \Pr[w_v \geq w] = \frac{\lambda w^2}{2^d n} \cdot w^{-(\tau-1)} = \frac{\lambda w^{3-\tau}}{2^d n}.$$

Consequently, the expected number of vertices of weight at least $w$ in $C'$ is

$$\mathbb{E}\left[|\{v \in V(C') \mid w_v \geq w\}|\right] = \frac{\lambda w^{3-\tau}}{2^d}.$$

Since the vertices are distributed according to a Poisson distribution, the probability for $C'$ to not contain any of these vertices is given by

$$\Pr[\{v \in V(C') \mid w_v \geq w\} = \emptyset] = \exp\left(-\frac{\lambda w^{3-\tau}}{2^d}\right).$$

Finally, we lower-bound the probability for the vertices of weight at least $w$ in our initial cell $C$ to induce a connected graph, by considering the probability that none of its sub-cells is empty. Note that we have

$$k = \left(\left\lfloor \frac{\Delta}{\Delta_w} \right\rfloor\right)^d$$

sub-cells $C'_1, \dots, C'_k$ that are *completely* contained in the cell $C$. Clearly, whether the remaining sub-cells (intersecting the boundary of $C$) are empty or not has no impact on the connectedness of the considered subgraph. The probability for all of the sub-cells $C'_1, \dots, C'_k$ to be non-empty can be simplified by applying union bound, which yields

$$\begin{aligned}\Pr[\forall C' \in \{C'_1, \dots, C'_k\} \colon V(C') \neq \emptyset] &= (1 - \Pr[V(C') = \emptyset])^k \\ &\geq 1 - k \cdot \Pr[V(C') = \emptyset] \\ &= 1 - \left(\left\lfloor \frac{\Delta}{\Delta_w} \right\rfloor\right)^d \cdot \exp\left(-\frac{\lambda w^{3-\tau}}{2^d}\right) \\ &\geq 1 - \Delta^d \cdot \frac{2^d n}{\lambda w^2} \cdot \exp\left(-\frac{\lambda w^{3-\tau}}{2^d}\right) \\ &= 1 - \frac{(2\Delta)^d}{\lambda w^2} \cdot \exp\left(-\frac{\lambda w^{3-\tau}}{2^d}\right) \cdot n. \end{aligned}$$

◀

The following lemma shows that we basically get an independent coin-flip with non-vanishing success probability for each cell to be nice. We want to point out three technical details of the lemma statement here. First, the lemma specifically considers the connected component containing a vertex of weight at least $\hat{w}$. We will later choose $\hat{w} = \sqrt{n/\lambda}$, i.e., this vertex is part of the core. As all core vertices form a clique, this makes sure that the components we get for the individual cells actually connect to one large component in the whole graph. Secondly, the lower bound on $\mu$, which is the expected number of vertices in the cell, given by $\mu = \Delta^d n$, requires that the cells are sufficiently large to contain a vertex of weight $\hat{w}$ with non-vanishing probability. Thirdly, the lower bound on $\hat{w}$ ensures that vertices with higher weight are likely connected by Lemma 2.

▶ **Lemma 3.** *Let $G \sim \mathcal{G}(n, \mathbb{B}^d, \tau, \lambda)$ be a threshold GIRG, let $\hat{w}$ be a weight, and let $C$ be a cell of side length $\Delta$ containing $\mu$ vertices in expectation. Then, with non-vanishing probability, the graph induced by the vertices in $C$ contains a vertex of weight at least $\hat{w}$ whose connected component has size $\Theta(\mu)$, if $\mu \geq \hat{w}^{\tau-1}$, $\mu \in \omega((\log n)^{2/(3-\tau)} \log\log(n)^d)$, and $\hat{w} \in \omega((\log n)^{1/(3-\tau)})$.*

**Proof.** The overall argument goes as follows. First, the lower bound on $\mu$ ensures that $C$ contains a vertex of weight $\hat{w}$ with non-vanishing probability. For a smaller weight $\overline{w} \leq \hat{w}$, we then apply Lemma 2 to get that all vertices of weight at least $\overline{w}$ form a connected component asymptotically almost surely. Afterwards, it remains to show that enough vertices of lower weight connect to a vertex of weight at least $\overline{w}$ via paths not leaving $C$. For the existence of these paths, we use Lemma 1. To show that most of them do not leave $C$, we use that the considered vertices have weight at most $\overline{w}$ and thus cannot deviate too much from the starting position.

Recall that the weight of a vertex is at least $\hat{w}$ with probability $\hat{w}^{-(\tau-1)}$. Thus, the expected number of vertices in cell $C$ with weight at least $\hat{w}$ is $\mu\hat{w}^{-(\tau-1)}$. Plugging in the bound $\mu \geq \hat{w}^{\tau-1}$, everything cancels and we obtain an expected value of 1. As the number of vertices in $C$ with weight above $\hat{w}$ follows a Poisson distribution, we get at least one such vertex with non-vanishing probability.

We set $\overline{w} = ((2^d/\lambda)\log n)^{1/(3-\tau)}$. Note that by the condition on $\hat{w}$ in the lemma statement, we have $\overline{w} \leq \hat{w}$ for sufficiently large $n$. Note further that $\overline{w}$ is chosen such that the exponent in the bound of Lemma 2 simplifies to $-\log n$. Thus by Lemma 2, the graph induced by the vertices of weight at least $\overline{w}$ in $C$ is connected with probability at least $1 - (2\Delta)^d/(\lambda\overline{w}^2)$. As $\Delta \leq 1$ and $\overline{w}$ is increasing with $n$, this goes to 1 for $n \to \infty$.

Consider a vertex of weight below $\overline{w}$. Then, by Lemma 1, it has a layer path to a vertex with weight at least $\overline{w}$ with non-vanishing probability. In the following, with *layer path* we always refer to a layer path that ends in the layer belonging to $\overline{w}$. Note that a layer path

has length at most $O(\log\log n)$. Also note that the largest weight we encounter is in $O(\overline{w})$ as the path stops in the layer belonging to weight $\overline{w}$ and the weights increase only by a constant factor between layers. It follows that, in each dimension, the distance between two consecutive vertices on a layer path is in $O((\overline{w}^2/n)^{1/d})$, as the vertices would not be connected otherwise. Thus, the overall deviation of a layer path from the starting point is upper bonded by $O((\overline{w}^2/n)^{1/d}\log\log n) = O\left(((\log n)^{2/(3-\tau)}\log\log(n)^d/n)^{1/d}\right)$. By the second lower bound on $\mu$, this is asymptotically less than $\Delta$. Thus, shrinking $C$ accordingly from all directions yields a subregion $C'$ that contains $\Theta(\mu)$ vertices in expectation such that any layer path that starts in $C'$ stays in $C$.

Instead of counting all vertices in $C'$ that have layer paths, we only count vertices in the first layer. This has the advantage, that the event that an individual vertex in the first layer has a layer path is independent of the number of vertices in the first layer (while it depends on the number of vertices in higher layers). First note that the number of vertices in the first level of $C'$ is a random variable following a Poisson distribution with expected value in $\Theta(\mu)$. Thus, there are $\Theta(\mu)$ such vertices with non-vanishing probability.

Now let $X \in [0,1]$ be the random variable that describes the fraction of vertices in the first layer that fail to have a layer path. By Lemma 1, the probability for an individual vertex to not have a layer path is a upper bounded constant $p < 1$ (i.e., the layer path exists with non-vanishing probability at least $1-p$). Thus, we get $\mathbb{E}[X] \leq p$. Markov's inequality then gives us $\Pr[X \geq c] \leq p/c$ and thus $\Pr[X < c] \geq 1 - p/c$. We can choose $c$ to be a constant with $p < c < 1$, which gives us a non-vanishing probability that a fraction of at least $1 - c > 0$ vertices have the desired layer path. Note that this holds independently of the number of vertices actually sampled in the first layer of $C'$.

To wrap up, consider the three events that there exists a vertex of weight at least $\hat{w}$, that there are $\Theta(\mu)$ vertices in the first layer of $C'$, and that a constant fraction of them have layer paths. Note that the three events are independent and each holds with non-vanishing probability. Thus, their intersection, which we denote with $A$, also holds with non-vanishing probability. Finally, the event $B$ that all vertices of weight at least $\overline{w}$ induce a connected graph holds asymptotically almost surely. Though $A$ and $B$ are not independent, we can apply the union bound to their complements to obtain that $A$ and $B$ together hold with non-vanishing probability. ◀

## 3.3 Large Components are Likely to Exist

To obtain the following theorem, it remains to apply a Chernoff bound to the coin flips obtained for each cell by Lemma 3.

▶ **Theorem 4.** *Let $G \sim \mathcal{G}(n, \mathbb{B}^d, \tau, \lambda)$ be a threshold GIRG. Then $G$ has a connected component of size $\Theta(n)$ with probability $1 - \exp\left(-\Omega(n^{(3-\tau)/2})\right)$.*

**Proof.** First note that the probability to have $\omega(n)$ vertices is exponentially small and thus we only have to show the lower bound on the size of the largest connected component. To apply Lemma 3, we choose the cell width $\Delta$ such that $\Delta^d n = \mu = \hat{w}^{\tau-1}$ where we set $\hat{w} = \sqrt{n/\lambda}$. With this, we obtain that the number of cells $k$ is

$$k \in \Theta\left(\frac{1}{\Delta^d}\right) = \Theta\left(\frac{n}{\hat{w}^{\tau-1}}\right) = \Theta\left(\frac{n}{\left(\sqrt{n/\lambda}\right)^{\tau-1}}\right) = \Theta\left(n^{\frac{3-\tau}{2}}\right).$$

Note that this bound is valid even if $\Delta$ does not divide the ground space evenly. Further note that the chosen $\Delta$ satisfies the conditions of Lemma 3, the graph induced by each cell contains a vertex from the core whose connected component has size $\Theta(\mu)$ with non-vanishing probability. If this holds for a constant fraction of cells, we get a giant component, as all vertices of weight at least $\hat{w}$ form a clique in $G$. Thus, we have $k$ independent coin flips, each succeeding with a probability of $p > 0$, and we are interested in the number of successes $X$. To show that $X \in \Theta(k)$ is highly likely, we can simply apply a Chernoff bound (see [24, Theorem 4.4]). For $\delta \geq 0$, we get

$$\Pr\left[X \leq (1-\delta)\mathbb{E}[X]\right] \leq \exp\left(-\frac{\delta^2}{2}\mathbb{E}[X]\right).$$

As $\mathbb{E}[X] \in \Theta(k)$, this implies $\Pr\left[X \in o(k)\right] \leq \exp\left(-\Omega(k)\right)$. Inserting $k$ yields the claim. ◀

As already mentioned in Section 2, this directly implies the following corollary.

▶ **Corollary 5.** *Theorem 4 also holds with the torus $\mathbb{T}$ as ground space.*

The following lemma states a well known property of GIRGs, see e.g. [19, Lemma 3.12] and [22, Definition 2.8]. For the sake of transparency, we give a simple proof based on the notation established throughout the paper.

▶ **Lemma 6** (folklore)**.** *Let $H \sim \mathcal{G}(n, \mathbb{B}^d, \tau, \lambda, T)$ be a GIRG and let $G$ be the subgraph induced by the vertices within a cell of side length $\Delta = (f(n)/n)^{1/d}$. Then $G \sim \mathcal{G}(f(n), \mathbb{B}^d, \tau, \lambda, T)$.*

**Proof.** Note that we basically consider two ways to generate a graph and claim that they give the same probability distribution over graphs. Intuitively, this can be seen by generating points with weights in the cell $[0, \Delta]^d$, scaling it to the full ground space $[0, 1]^d$, and making three observations. First, for the vertex positions, this is equivalent to directly sampling points in $[0, 1]^d$. Secondly, the weight distribution is independent of the number of vertices. Thirdly, the connection probabilities between vertices are the same in the scaled variant as they are in the cell. To make this more formal, draw $G$ as a subgraph of $H$ as stated in the lemma and draw $G' \sim \mathcal{G}(f(n), \mathbb{B}^d, \tau, \lambda, T)$. We show that $G$ and $G'$ follow the same distribution.

Recall that we consider the Poisson variant of the GIRG model, i.e., the vertices are the result of a Poisson point process in the product space $\mathbb{B}^d \times \mathcal{W}$. Thus, the vertex set of $G$ can be generated by first determining the number of points $n_G$ with positions in $[0, \Delta]^d$, which is a random variable following a Poisson distribution with expectation $n \cdot \Delta^d = f(n)$. Then, independently for each of the $n_G$ vertices, a position is drawn uniformly at random from $[0, \Delta]^d$ and a weight is drawn from $(1, \infty)$ with probability density function $(\tau - 1) \cdot w^{-\tau}$.

To generate $G'$, we can also first determine the number of points $n_{G'}$, which is also Poisson distributed with expectation $f(n)$. Thus, we can couple $n_G$ and $n_{G'}$ to have the same value and we assume a one-to-one correspondence between the vertices in $G$ and $G'$ in the following. For each vertex, the weight is again a random variable with density $(\tau - 1) \cdot w^{-\tau}$, which only depends on $\tau$. Thus, for each vertex, we can couple its weight in $G$ with its weight in $G'$ to assume them to be equal. The position in $G'$ is drawn uniformly from $[0, 1]^d$. Thus, we can couple the random variables for the positions in $G'$ with those in $G$ such that a vertex with position $x \in [0, \Delta]^d$ in $G$ has position $x/\Delta$ in $G'$. Note that this has the effect that all distances between vertices in $G'$ are scaled by a factor of $1/\Delta$ compared to the corresponding distance in $G$.

It remains to show that for every vertex pair $u, v$ the connection probability in $G$ is the same as in $G'$. Let $w_u$ and $w_v$ be the weight of $u$ and $v$ (which is the same for $G$ and $G'$ due to the coupling). Also, let $\text{dist}(u, v)$ be the distance between $u$ and $v$ in $G$ and let $\text{dist}'(u, v) = \text{dist}(u, v)/\Delta$ be their distance in $G'$. Then (for $T > 0$) the connection probability of $u$ and $v$ in $G$ is

$$\Pr[\{u, v\} \in E] = \min\left\{\left(\frac{\lambda w_u w_v}{n \,\text{dist}(u, v)^d}\right)^{1/T}, 1\right\}.$$

The two things that change for $G'$ is that $n$ is replaced by $f(n)$ and $\text{dist}(u, v)^d$ is replaced by $\text{dist}'(u, v)^d = (\text{dist}(u, v)/\Delta)^d = n/f(n) \cdot \text{dist}(u, v)^d$. The $f(n)$ cancels out, yielding the same connection probability for $G$ and $G'$. For $T = 0$, the argument works analogously. ◀

Together with Theorem 4 this yields the following corollary. We note that this also yields large connected components within cells that are too small to contain a core vertex. For such cells, we know that we get a large connected component but we do not know whether it connects to the giant of the whole graph. Clearly, the same statement holds with the torus $\mathbb{T}^d$ as ground space.

▶ **Corollary 7.** *Let $H \sim \mathcal{G}(n, \mathbb{B}^d, \tau, \lambda)$ be a threshold GIRG and let $G$ be the subgraph induced by the vertices within a cell of side length $\Delta = (f(n)/n)^{1/d}$. Then $G$ has a connected component of size $\Theta(f(n))$ with probability $1 - \exp\left(-\Omega(f(n)^{(3-\tau)/2})\right)$.*

## 4 Conclusion

Our proof for the emergence of a giant component in geometric inhomogeneous random graphs builds on three simple arguments. First, GIRGs are likely to contain a clique of high-weight vertices. Second, the remaining vertices are sufficiently likely to connect to this core via layer-paths, whose vertices have exponentially increasing weight. And, third, most of these paths exist sufficiently independently from each other.

We note that the same argumentation also works for the closely related hyperbolic random graph model, where the discretization into weight layers translates to a natural discretization of the underlying geometric space that was previously used to bound the diameter of HRGs [15].

Our resulting strong probability bound can be combined with a simple coupling argument to identify connected subgraphs of arbitrary size in certain subregions of the geometric ground space. In particular, when these subregions are the cells of a regular grid (as used several times throughout the paper), we obtain connected subgraphs of roughly equal size. We believe that this property can be utilized in the context of problems with connectivity constraints. For example, in the previously mentioned *balanced connected partitioning* problem [8, 27], the goal is to partition the vertices of a graph into a given number of sets of approximately equal size, such that their induced subgraphs are connected. Moreover, in *component order connectivity* [16] the aim is to find a minimum number of vertices such that after their removal each connected component has bounded size. We conjecture that our structural insights in Corollary 7 may prove useful in obtaining efficient algorithms for these problems on GIRGs and the networks they represent well.

## References


1 William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001. doi:10.1080/10586458.2001.10504428.

2 Martin J.B. Appel and Ralph P. Russo. The connectivity of a graph on uniform points on $[0,1]^d$. *Statistics & Probability Letters*, 60(4):351–357, 2002. doi:10.1016/S0167-7152(02)00233-X.

3 Thomas Bläsius and Philipp Fischbeck. On the External Validity of Average-Case Analyses of Graph Algorithms. In *European Symposium on Algorithms (ESA 2022)*, pages 21:1–21:14, 2022. doi:10.4230/LIPIcs.ESA.2022.21.

4 Thomas Bläsius, Tobias Friedrich, and Christopher Weyand. Efficiently Computing Maximum Flows in Scale-Free Networks. In *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204, pages 21:1–21:14, 2021. doi:10.4230/LIPIcs.ESA.2021.21.

5 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. Efficiently generating geometric inhomogeneous and hyperbolic random graphs. *Network Science*, 10(4):361–380, 2022. doi:10.1017/nws.2022.32.

6 Michel Bode, N. Fountoulakis, and Tobias Müller. On the largest component of a hyperbolic model of complex networks. *Electronic Journal of Combinatorics*, 22:1–46, August 2015. doi:10.1214/17-AAP1314.

7 Karl Bringmann, Ralph Keusch, Johannes Lengler, Yannic Maus, and Anisur R. Molla. Greedy routing and the algorithmic small-world phenomenon. *Journal of Computer and System Sciences*, 125:59–105, 2022. doi:10.1016/j.jcss.2021.11.003.

8 Yong Chen, Zhi-Zhong Chen, Guohui Lin, Yao Xu, and An Zhang. Approximation Algorithms for Maximally Balanced Connected Graph Partition. *Algorithmica*, 83:3715–3740, 2021. doi:10.1007/s00453-021-00870-3.

9 Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002. doi:10.1073/pnas.252631999.

10 Fan Chung and Linyuan Lu. Connected Components in Random Graphs with Given Expected Degree Sequences. *Annals of Combinatorics*, 6(2):125–145, 2002. doi:10.1007/PL00012580.

11 Josep Díaz, Dieter Mitsche, and Xavier Pérez-Giménez. On the Connectivity of Dynamic Random Geometric Graphs. In *Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 601–610, 2008. URL: https://dl.acm.org/doi/abs/10.5555/1347082.1347149.

12 Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

13 Paul Erdős and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

14 Nikolaos Fountoulakis and Tobias Müller. Law of large numbers for the largest component in a hyperbolic model of complex networks. *The Annals of Applied Probability*, 28(1):607–650, 2018. URL: https://www.jstor.org/stable/26542317.

15 Tobias Friedrich and Anton Krohmer. On the Diameter of Hyperbolic Random Graphs. *SIAM Journal on Discrete Mathematics*, 32(2):1314–1334, 2018. doi:10.1137/17M1123961.

16 Daniel Gross, Monika Heinig, L. Iswara, L.W. Kazmierczak, Kristi Luttrell, John Saccoman, and C. Suffel. A Survey of Component Order Connectivity Models of Graph Theoretic Networks. *WSEAS Transactions on Mathematics*, 12:895–910, 2013.

17 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random Hyperbolic Graphs: Degree Sequence and Clustering. In *39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 573–585, 2012. doi:10.1007/978-3-642-31585-5_51.

18 P. Gupta and P.R. Kumar. Critical power for asymptotic connectivity. In *37th IEEE Conference on Decision and Control*, volume 1, pages 1106–1110, 1998. doi:10.1109/CDC.1998.760846.

19 Ralph Keusch. *Geometric Inhomogeneous Random Graphs and Graph Coloring Games*. PhD thesis, ETH Zurich, 2018. doi:10.3929/ethz-b-000269658.

20 Marcos Kiwi and Dieter Mitsche. On the second largest component of random hyperbolic graphs. *SIAM Journal on Discrete Mathematics*, 33(4):2200–2217, 2019. doi:10.1137/18M121201X.

21 Júlia Komjáthy, John Lapinskas, and Johannes Lengler. Penalising transmission to hubs in scale-free spatial random graphs. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 57(4):1968–2016, 2021. doi:10.1214/21-AIHP1149.

22 Júlia Komjáthy and Bas Lodewijks. Explosion in weighted hyperbolic random graphs and geometric inhomogeneous random graphs. *Stochastic Processes and their Applications*, 130(3):1309–1367, 2020. doi:10.1016/j.spa.2019.04.014.

23 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82:036106, 2010. doi:10.1103/PhysRevE.82.036106.

24 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

25 Tobias Müller and Merlijn Staps. The Diameter of KPKVB Random Graphs. *Advances in Applied Probability*, 51(2):358–377, 2019. doi:10.1017/apr.2019.23.

26 Mathew Penrose. *Random Geometric Graphs*. Oxford University Press, 2003.

27 Stephan Schwartz. An overview of graph covering and partitioning. *Discrete Mathematics*, 345(8):112884, 2022. doi:10.1016/j.disc.2022.112884.

28 Bo Söderberg. General formalism for inhomogeneous random graphs. *Phys. Rev. E*, 66:066121, 2002. doi:10.1103/PhysRevE.66.066121.

# An Efficient Algorithm for Power Dominating Set

**Thomas Bläsius**
Karlsruhe Institute of Technology (KIT), Germany

**Max Göttlicher**
Karlsruhe Institute of Technology (KIT), Germany

## Abstract

The problem Power Dominating Set (PDS) is motivated by the placement of phasor measurement units to monitor electrical networks. It asks for a minimum set of vertices in a graph that observes all remaining vertices by exhaustively applying two observation rules. Our contribution is twofold. First, we determine the parameterized complexity of PDS by proving it is $W[P]$-complete when parameterized with respect to the solution size. We note that it was only known to be $W[2]$-hard before. Our second and main contribution is a new algorithm for PDS that efficiently solves practical instances.

Our algorithm consists of two complementary parts. The first is a set of reduction rules for PDS that can also be used in conjunction with previously existing algorithms. The second is an algorithm for solving the remaining kernel based on the implicit hitting set approach. Our evaluation on a set of power grid instances from the literature shows that our solver outperforms previous state-of-the-art solvers for PDS by more than one order of magnitude on average. Furthermore, our algorithm can solve previously unsolved instances of continental scale within a few minutes.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Power Dominating Set, Implicit Hitting Set, Parameterized Complexity, Reduction Rules

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.21

**Related Version** *Full Version*: `https://arxiv.org/abs/2306.09870` [4]

**Supplementary Material** *Software (Source Code)*: `https://gitlab.com/Aldorn/pds-code` archived at `swh:1:rev:121131be58bc1b6f91a4863bd01fe7cbd0439ab9`

**Funding** *Max Göttlicher*: German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation.

## 1 Introduction

Monitoring power voltages and currents in electric grids is vital for maintaining their stability and for cost-effective operation. The sensors required to obtain high-resolution measurements, so-called phasor measurement units, are expensive pieces of equipment. The goal to place as few of those sensors as possible to minimize cost is called the Power Dominating Set problem (PDS). It was first posed by Mili, Baldwin and Adapa. [16] and formalized by Baldwin et al. [2]. In its basic form, the problem asks whether the graph of a power grid can be observed by exhaustively applying two observation rules [7]: First, every sensor observes its vertex and all neighbors. Secondly, if a vertex is observed and has only one unobserved neighbor, that neighbor becomes observed, too.

PDS is unfortunately NP-complete [7, 11, 14], i.e., we cannot expect there to be an algorithm that performs reasonably on all inputs. Moreover, the problem remains hard for a wide range of different graph classes [7, 8, 11, 14, 20, 9, 15]. In terms of parameterized complexity, PDS is known to be $W[2]$-hard [9] when parameterized by solution size.


On the positive side, various approaches for solving PDS have been proposed. Theoretic results show that PDS can be solved in linear time in graphs with fixed tree-width [14, 9]. However, those algorithms have, to the best of our knowledge, never been implemented and are probably infeasible in practice due to their bad scaling with respect to the tree-width. Several exponential-time algorithms have been presented [7, 3] but those algorithms have not been implemented and evaluated either.

Practically feasible approaches using an MILP formulation have been proposed by Aazami [1]. This formulation was later improved upon by Brimkov, Mikesell, and Smith [6] and most recently Jovanovic and Voss [13]. A different approach is to reduce PDS to the hitting set problem [5, 18]. This approach is based on the observation that one can determine so-called *forts*, which are subsets of vertices that prevent propagation if none of them is selected. A set of vertices is a valid solution for PDS if and only if at least one vertex is selected for each fort, i.e., if it is a hitting set for the collection of all forts. Graphs may contain an exponential number of forts, so this hitting set instance is not computed explicitly. Instead, one can use the so-called *implicit hitting set* approach, where one starts with a subset of all forts, computes a hitting set for this subset, and then validates whether this is already a solution for the PDS instance. If not, one obtains at least one new fort that can be added to the set of considered forts. This is iterated until a solution is found. This implicit hitting set approach has been used for other problems, e.g., for MaxSAT [17] and TQBF [12]. For PDS, it has been introduced by Bozeman et al. [5]. The strategy of finding forts has been later improved by Smith and Hicks [18], providing the current state-of-the-art for solving PDS in practice.

Our contribution is threefold. First, we study the parameterized complexity of PDS parameterized by the solution size. Though it is known to be $W[2]$-hard [9], it was unknown whether PDS is also contained in $W[2]$. We show that PDS is $W[P]$-complete via a reduction from Weighted Circuit Satisfiability for circuits of arbitrary weft. This completely determines its parameterized complexity and in particular shows that it is not in $W[2]$ unless $W[2] = W[P]$. In our second contribution, we propose a set of reduction rules for pre-processing PDS instances. Our reduction rules aim to produce equivalent instances that are smaller and annotated with partial decisions, i.e., some vertices are marked as selected or as forbidden-to-select. Though these annotations lead to a more general problem than the basic PDS, we show that existing approaches for solving PDS can be easily adapted to solve the annotated instances. Moreover, we show that their performance greatly benefits from our reduction rules. Finally, our third contribution is an improved heuristic for finding forts for the implicit hitting set formulation. This improved heuristic together with our reduction rules beats the current state of the art solvers by more than one order of magnitude. Moreover, our approach can solve previously unsolved instances of continental scale.

The remainder of this paper is organized as follows. Section 2 provides an overview of the basic concepts and notation used throughout this paper. In Section 3, we show that PDS is $W[P]$-complete. Our reduction rules and the heuristic for extending the hitting set instance are presented in Section 4. Section 5 contains our experimental evaluation of the new method using a set of benchmark instances.

## 2 Preliminaries

**Graphs and Neighborhoods.** Let $G = (V, E)$ be an undirected graph with vertices $V$ and edges $E$. For $v \in V$, let $N(v) = \{u \in V \mid uv \in E\}$ be the *open neighborhood* of $v$. Similarly, $N[v] = N(V) \cup \{v\}$ is the *closed neighborhood* of $v$. Given a set $S \subseteq V$ we denote by $N(S)$ and $N[S]$ the union of all open and closed neighborhoods of the vertices in $S$.

**Power Dominating Set.** For a given graph $G$, the problem *Power Dominating Set (PDS)* is to find a minimum vertex set $S \subseteq V$ of *selected* vertices such that all vertices of the graph are observed. We call such set a *power dominating set*. The size of a minimum power dominating set of a given graph $G$ is called the *power dominating number* $\gamma_P(G)$. Whether a vertex is *observed* is determined by the following rules, which are applied iteratively. We note that for the second rule, vertices can be marked as *propagating*, i.e., the input of PDS is not just a graph but a graph together with a set of propagating vertices.

**Domination rule.** A vertex is observed if it is in the closed neighborhood of a selected vertex.

**Propagation rule.** Let $u \in V$ be a propagating vertex. If $u$ is observed and $v \in N(u)$ is the only neighbor of $u$ that is not yet observed, then $v$ becomes observed[1]. If the propagation rule is applied to an observed vertex $u$, we say it *propagates* its observation status.

The special case where we have no propagating vertices yields the well known *Dominating Set (DS)* problem. Moreover, the special case where all vertices are propagating is called *simple-PDS*. In addition to the above Dominating Set variants, we also consider the extension variant Dominating Set Extension. For DS-Extension, the input consists of the graph $G = (V, E)$, a set $X \subseteq V$ of *pre-selected* and a set $Y$ of *excluded* vertices; vertices in $V \setminus X \setminus Y$ are called *undecided*. DS-Extension asks whether there exists a solution $S \subset V$ such that $S$ includes all selected and excludes all excluded vertices, i.e., $X \subseteq S$ and $Y \cap S = \emptyset$. The problems PDS-Extension and simple-IPDS-Extension are defined analogously.

**Hitting Set.** Let $V$ be a set and let $\mathcal{F} \subseteq 2^V$ be a family of subsets. A set $H \subseteq V$ is a *hitting set* if it *hits* every set $F \in \mathcal{F}$, i.e., $F \cap H \neq \emptyset$ for all $F \in \mathcal{F}$. The problem *Hitting Set* is to find a hitting set of minimum size. Note that the extension variant of Hitting Set reduces to an instance of Hitting Set itself, as one can simply remove excluded elements and remove the sets containing pre-selected elements.

## 3 Power Dominating Set is $W[P]$-Complete

We prove $W[P]$-completeness via a chain of parameterized reductions from the Weighted Monotone Circuit Satisfiability (WMCS) problem. WMCS has a monotone Boolean circuit as input and asks whether it can be satisfied by setting at most $k$ inputs to true, where $k$ is the parameter. We assume familiarity with the $W$-hierarchy and parameterized reductions.We start by introducing a variant of the PDS problem that we use as an intermediate problem in our chain of reductions. For brevity, we only sketch an outline of the proof in this section; for the full proof see [4].

The input of the problem *Implicating Power Dominating Set (IPDS)* is an instance of PDS with the following additional information. First, edges of the graph $G = (V, E)$ can be marked as *booster edges*. Secondly, we are given a set of *implication arcs* $A \subseteq V \times V$. We interpret $A$ as a set of directed edges on $V$ but perceive them as separate from the graph $G$, i.e., they do not affect the neighborhood. In addition to the domination and propagation rule introduced in Section 2, we define the following to observation rules.

**Booster rule.** Let $uv \in E$ be a booster edge. If $u$ is observed, then $v$ becomes observed and vice versa.

**Implication rule.** Let $(u, v) \in A$ be an implication arc and let $u$ be observed. Then $v$ also becomes observed.

[1] The propagation rule is motivated by Kirchoff's law and Ohm's law in electric networks. Propagating vertices are also called *zero-injection vertices*. In electric networks, they refer to buses in substations that have no attached loads.

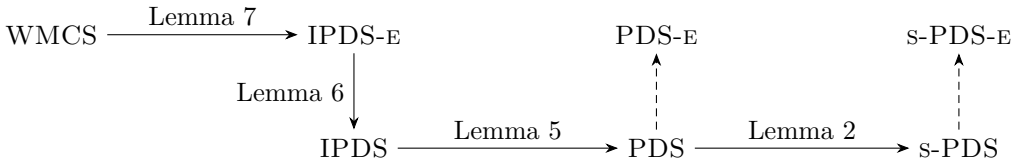



**Figure 1** Reduction steps to show that PDS and its variants are $W[P]$ hard. The solid arrows indicate our parameterized reductions described in this section. The hardness of the extension problems follows from the hardness of their basic problem, indicated by the dashed arrows.

We note that IPDS is a generalization of PDS in the sense that every PDS instances is an instance of IPDS with no booster edges and an empty set of implication arcs. The extension variant IPDS-Extension is defined analogously to PDS-Extension. Proving containment of IPDS-Extension in $W[P]$ is straight forward by giving an appropriate non-deterministic Turing machine. We note that the analogous statement has been observed before for PDS by Kneis et al. [14]. Note that this implies containment in $W[P]$ for all other problem variants we defined.

▶ **Lemma 1.** *Implicating Power Dominating Set Extension is in $W[P]$.*

As all other variants of the power dominating set problem we consider are special cases of IPDS-Extension, this also proves containment in $W[P]$ for the other variants.

**Power Dominating Set to Simple Power Dominating Set.** Our chain of reductions to prove $W[P]$-hardness is illustrated in Figure 1. We start with the reduction from PDS to Simple PDS, which is similar to the proof of $W[2]$ hardness of PDS [14, 9]. The core idea is to simulate a non-propagating vertex with a propagating vertex with an additional leaf attached.

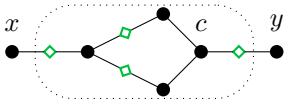
▶ **Lemma 2.** *There is a parameterized reduction from Power Dominating Set to Simple Power Dominating Set.*

**Proof Sketch.** Similar to the proof of $W[2]$ hardness of Power Dominating Set [14, 9], we can attach a leaf to each non-propagating vertex. Selecting the leaf as part of a solution is never optimal: one can instead choose its neighbor. Then, a vertex with an attached leaf can never propagate to any vertex except the leaf. ◀

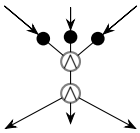
**Implicating Power Dominating Set to Power Dominating Set.** The reduction from IPDS to PDS, works in two steps. First, we show that we can eliminate implicating arcs by replacing each of them with the small gadget show in Figure 2a. Using another gadget, we eliminate booster edges in a similar way, yielding the reduction.

▶ **Lemma 3.** *Every instance of Implicating Power Dominating Set can be reduced to an equivalent instance with no implication arcs without changing the parameter.*

**Proof Sketch.** Given an IPDS-instance $G$, we replace all implication arcs $a = (x, y)$ with the gadget depicted in Figure 2a. To see why the gadget works as desired, consider the implication gadget and first assume that $x$ is observed. By applying the booster and the propagation rules, one can verify that all vertices introduced in the gadget and $y$ become observed. Conversely, if only $y$ is observed, $c$ becomes observed by the booster rule but cannot propagate due to its two unobserved neighbors. Thus, the gadget mimics the behavior of an implication arc. ◀




**(a)** Gadget simulating an implication arc from $x$ to $y$ using booster edges (marked with green diamonds).



**(b)** A gadget for simulating an `and` gate. The bottom $\wedge$-node is observed iff all input nodes are observed.

**Figure 2** Gadgets for implication arcs and `and` gates.

Our gadget for simulating booster edges requires adding a globally unique non-propagating vertex $b$ to which all such gadgets are connected. The gadget in turn replaces a booster edge between $x$ and $y$ with a new vertex $v_{xy}$ which is connected to $x$, $y$ and $b$. We enforce that $b$ is selected by attaching a leaf.

▶ **Lemma 4.** *Every IPDS-instances with booster edges can be reduced to an equivalent instance without booster edges.*

**Proof Sketch.** One can verify that the booster gadget works as intended as follows: by the domination rule, $b$ observes $v_{xy}$. If now either of $x$ or $y$ becomes observed, we can apply the propagation rule on $v_{xy}$, observing the other. Note that the inserted vertex $b$ is the same for all booster gadgets. ◀

▶ **Lemma 5.** *There is a parameterized reduction from* Implicating Power Dominating Set *to* Power Dominating Set.

**Extension to Non-Extension (for IPDS).** For the IPDS-Extension to IPDS, the core difficulty comes from enforcing the excluded vertices to not be selected. We already saw how to enforce the selection of vertices in the construction of the booster gadget. The basic idea for excluding vertices from the solution is to make the selection of certain vertices very expensive.

▶ **Lemma 6.** *There is a parameterized reduction from* Implicating Power Dominating Set Extension *to* Implicating Power Dominating Set.

**Proof Sketch (Vertex Exclusion).** Let $G = (V, E)$ be an IPDS-Extension-instance where the vertices $Y \subseteq V$ are excluded from a solution. We construct an equivalent instance without excluded vertices. We achieve this by creating a new graph $G'$, consisting of $|V| + 1$ copies of $G$ and a clique $C$ of $|V \setminus Y|$ non-propagating vertices. Each copy $G^{(i)}$ has a fresh set of vertices representing the vertices in $G$ and there is an edge between two vertices in a copy if there is an edge between their counterparts in $G$. There are no edges between vertices in different copies. The vertices in $C$ represent the vertices not excluded from a solution. For each vertex in $C$ we add edges to the closed neighborhoods of their counterparts in each of the copies.

Given a power dominating set $S$ of $G$, selecting the corresponding vertices in $C$ yields a power dominating set for $G'$. Conversely, the construction ensures that vertices selected in one of the copies never observe any vertices in another copy. Thus, and because all copies are identical, if a minimum power dominating set contains a vertex in one of the copies, it must contain a vertex in all other copies, too. Hence, a minimum power dominating set of size less then $|V|$ cannot contain any vertices outside $C$. As the vertices in $C$ correspond to the selectable vertices in $G$, we obtain a power dominating set of $G$. ◀

**WMCS to IPDS-Extension.** Finally, for the reduction from WMCS to IPDS-Extension, the core idea is to replace the arcs in the directed acyclic graph describing the circuit with implication arcs and to model `and`-gates as show in Figure 2b.

▶ **Lemma 7.** *There is a parameterized reduction from Weighted Monotone Circuit Satisfiability to Implicating Power Dominating Set Extension.*

**Proof Sketch.** We construct an equivalent IPDS-Extension-instance $G$ from a given monotone circuit $C = (V, E)$ as follows. In this construction, we interpret TRUE values in the circuit as a node being observed. We thus interpret all directed edges in the circuit as implication arcs and add further implication arcs from the output to every input. The input nodes become propagating vertices, all other vertices in $G$ are non-propagating. The `or`-gates are simulated by the implication rule without further adaptation.

To simulate the `and`-gates, we use the gadget in Figure 2b. We replace every `and`-gate $v$ by two new connected vertices $x$ and $y$ where all outgoing edges of $v$ are instead outgoing implication arcs of $y$. For every incoming edge of $v$ from $u$, we place a new proxy vertex $x_u$ and add an edge $x_u x$ and an implication arc $(u, x_u)$. Then, the gate output $y$ becomes observed by the propagation rule from $x$ if and only if all proxy vertices $x_u$ are observed, i.e. if all inputs of the `and`-gate are TRUE.

We need the implication arcs back from the output to the inputs to ensure all vertices become observed if the corresponding truth assignment is satisfying. ◀

▶ **Corollary 8.** *Power Dominating Set is W[P] complete.*

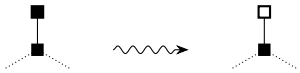
## 4 Solving Power Dominating Set

In this section, we give an algorithm for solving PDS-Extension. Our algorithm consists of different phases. In the first phase, we apply the reduction rules described in Section 4.1. Each rule either shrinks the graph or decides for a vertex that it should be pre-selected or excluded. We prove that the rules are safe, i.e., they yield equivalent instances. Afterwards, in Section 4.2 we split the instance into several components that can be solved independently. Finally, each of the subinstances is solved exactly using the implicit hitting set approach [5] with our improved strategy for finding new sets that need to be hit; see Section 4.3.

We note that these phases are somewhat modular in the sense that one could easily add further reduction rules or that one can replace the algorithm for solving the kernel in the final step. In our experiments in Section 5, we also use an MILP for this step. This MILP formulation is based on the formulation for PDS by Jovanovic and Voss [13] with the adjustments from [4, Appendix A]. Moreover, instead of solving the subinstances optimally, one can instead use a heuristic solver. The preceding application of our reduction rules and splitting into subinstances then helps to find better solutions rather than improving the running time of exact solvers. This is used in our experiments to find upper bounds on the power domination number before we have an exact solution.

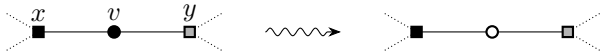
### 4.1 Reduction Rules

Many of our reduction rules are local in the sense that they transform one substructure into a different substructure. Most of our local reduction rules are illustrated in Figure 3. In the following, we specify additional reduction rules that are either non-local or otherwise difficult to illustrate. For more details and proofs of safeness, see [4, Appendix B].

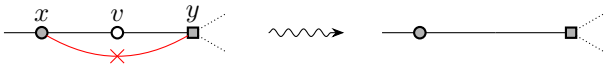


**(a)** `Deg1a`: If an undecided leaf is attached to a non-excluded vertex, the reduction rule excludes the leaf.

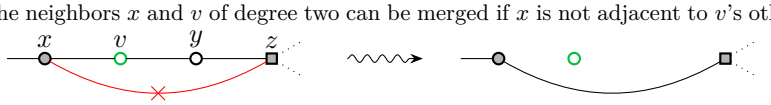
**(b)** `Deg1b`: Two cases of excluded leaves attached to a vertex. On the left, the parent vertex is propagating and becomes non-propagating. On the right, the parent vertex is non-propagating and becomes pre-selected. The attached leaf is removed in both cases.



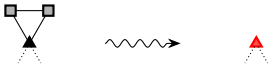

**(c)** `Deg2a`: A vertex $v$ of degree two can safely be excluded if it has an undecided neighbor.




**(d)** `Deg2b`: The neighbors $x$ and $v$ of degree two can be merged if $x$ is not adjacent to $v$'s other neighbor $y$.

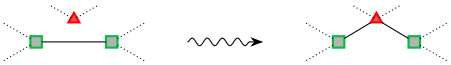



**(e)** `Deg2c`: If $v$ is observed and has two non-adjacent unobserved neighbors of degree two, $x$ and $y$, with $y$ being excluded, then we remove $y$ and the edge $xv$ and connect $x$ to the other neighbor $z$ of $y$.



**(f)** `Tri`: We pre-select the undecided and remove the two vertices of degree two in the triangle.

**(g)** `OnlyN`: If an excluded vertex is surrounded by non-propagating vertices, only one of which is not excluded, we pre-select the non-excluded vertex.



**(h)** `ObsE`: An edge with two observed endpoints is replaced by edges to an arbitrary pre-selected vertex. This leads to fewer applications of the propagation rule by replacing them with the domination rule. Together with `Deg1b`, `Deg2a` and `Deg2b` this also helps reduce the graph size.

**Figure 3** Illustrated overview of the local reduction rules. Round vertices ● are propagating, triangular vertices ▲ are non-propagating, square vertices ■ may be propagating or non-propagating. Hollow vertices □ are excluded from a solution, vertices filled black ■ are undecided, red triangular vertices ▲ are pre-selected. Vertices filled gray ■ may be undecided, excluded or pre-selected. Green vertices □ are observed but not pre-selected. The absence of an edge is indicated in red ⊣⊢.

▶ **Reduction** `ObsNP` (Observed Non-Propagating). *Let $v$ be an observed, non-propagating and excluded vertex. Then delete $v$.*

▶ **Reduction** `Isol` (Isolated). *Let $v$ be an undecided isolated vertex. Then pre-select $v$.*

For the next two reduction rules, we introduce the concept of *observation neighborhood*. For a set of vertices $U \subseteq V$ the observation neighborhood is the set of vertices that is observed when selecting $U$ in addition to all pre-selected vertices and applying the observation rules exhaustively. For convenience, we define the observation neighborhood of a single vertex $v$ to be the observation neighborhood of the single element set $\{v\}$.

▶ **Reduction** `Dom` (Domination). *If the closed neighborhood of some undecided vertex $w$ is contained in the observation neighborhood of some other undecided vertex $v$, exclude $w$.*

▶ **Reduction** `NecN` (Necessary Node). *Let $v$ be an undecided vertex. If the observation neighborhood of all undecided vertices except $v$ does not contain all vertices in $G$, select $v$.*

We note that Binkele-Raible and Fernau [3] already introduced reduction rules for their exponential-time algorithm. We do not use them in our algorithm as they are not generally applicable but rather require a specific situation. The only exception [3, "isolated"] is superseded by our reduction rules.

### 4.1.1 Order of Application

In a first step, use a depth first search and process the vertices in post-order to apply the rules `Deg1a`, `Deg1b`, and `Deg2a`. The order in which we process the vertices is important here, as it makes sure that attached paths are properly reduced. This is only relevant for this first application of the reduction rules and in later applications, we process the vertices and edges in arbitrary order. After this initial application, we iterate the following three steps until no reduction rules can be applied. (i) Iterate the application of the local reduction rules (`Deg1a`, `Deg1b`, `Deg2a`, `Deg2b`, `Tri`, `Deg2c`, `OnlyN`, `ObsNP`, `ObsE`) until no local reduction rule is applicable. (ii) Apply the non-local rule `Dom`. (iii) Apply the non-local rule `NecN`.

We note that applying `Dom` once to all vertices is exhaustive in the sense that it cannot be applied again immediately afterwards. It can, however, become applicable again after rerunning the other reduction rules. The same is true for `NecN`.

Our reasoning for this sequence of application is that the local rules are more efficient than the non-local ones. Thus, we first apply the cheap rules exhaustively before resorting to the expensive ones. Preliminary experiments showed that further tweaking the order of application has only minor effect on the kernel size and run time.

### 4.1.2 Implementation Notes

The naïve implementation of the reduction rules can be very slow, in particular for the non-local rules. The costly operation in those rules is the computation of the observation neighborhood. We thus use a specialized data structure that allows us to pre-select and deselect vertices in arbitrary order. Each time we pre-select a vertex, we also update the observed vertices and keep track of which vertex propagates to which other vertex. For de-selecting vertices, we only mark vertices as unobserved, that were directly or indirectly observed by the deselected vertex. Being able to select and deselect arbitrary vertices allows a straightforward implementation of the non-local rules.

## 4.2 Split into Subinstances

While the propagation rule may have non-local effects within the whole graph, propagation cannot pass through selected vertices. This is formalized by the following theorem.

▶ **Theorem 9.** *Let $G = (V, E)$ be the graph with pre-selected vertices $X \subseteq V$ and let $C_1, \ldots, C_\ell \subseteq V$ be the vertices in the connected components of the sub-graph of $G$ induced by $V \setminus X$. Further let $S_1, \ldots, S_\ell$ be minimum power dominating sets of the subgraphs induced by $N[C_1], \ldots, N[C_\ell]$. Then $S = S_1 \cup \cdots \cup S_\ell$ is a minimum power dominating set of $G$.*

These sub-problems can be identified in linear time using a depth-first search restarting at unexplored non-active nodes while ignoring outgoing edges of active nodes.

## 4.3 Solving the Kernel Via Implicit Hitting Set

We briefly describe the implicit hitting set approach. Compared to how Bozeman et al. [5] introduced it, we allow non-propagating vertices. However, this does not change any of the proofs and thus the approach directly translates to this slightly more general setting.

In a graph $G$, a *fort* is a non-empty subset of vertices $F \subseteq V(G)$ such that no propagating vertex outside $F$ is adjacent to precisely one vertex in $F$. A power dominating set must be a hitting set of the family of all *fort neighborhoods* in $G$, i.e., if $F$ is a fort and $S$ is a power dominating set, then $N[F] \cap S \neq \emptyset$. Conversely, if a hitting set for a family $\mathcal{F}$ of fort neighborhoods is not a power dominating set, then one can find an additional fort of $G$ whose neighborhood is not in $\mathcal{F}$.

This yields the following algorithm. Start with some set $\mathcal{F}$ of fort neighborhoods. Compute a minimum hitting set $H$ for $\mathcal{F}$. If $H$ is already a power dominating set, we have found the optimum. Otherwise, we construct at least one new fort neighborhood and add it to $\mathcal{F}$.

One core ingredient of this approach is the choice of which fort neighborhoods to add to $\mathcal{F}$. Previous approaches [18, 5] aimed at finding forts or fort neighborhoods that are as small as possible. The reasoning behind this is that the set of all fort neighborhoods can be exponentially large (even when restricted to those that are minimal with respect to inclusion) and thus it makes sense to add sets that are as restrictive as possible, hoping that only few sets suffice before the Hitting Set solution yields a PDS solution. However, finding forts of minimum size or minimum size fort neighborhoods is difficult while just finding any fort is easy. Moreover, if we add only few forts in every step, we have to potentially solve more Hitting Set instances. We thus propose to instead find multiple forts at once and to add them all to the Hitting Set instance. Our method of finding forts is based on the following lemma.

▶ **Lemma 10.** *Let $G = (V, E)$ be a graph and let $S \subseteq V$ be a set of selected vertices. Let further be $R$ the set of vertices observed by exhaustive application of the observation rules with respect to $S$. Then the set of unobserved vertices $V \setminus R$ is a fort.*

**Proof.** Assume $V \setminus R$ is not a fort. Then there exists a propagating vertex $v$ in $R$ that is adjacent to precisely one vertex $w$ in $V \setminus R$, i.e., $v$ has precisely one unobserved neighbor $w$. This constradicts the exhaustive application of the propagation rule and thus the set of unobserved vertices is a fort. ◀

By Lemma 10, whenever we have a candidate solution that does not yet observe all vertices, we obtain a new fort and can add its neighborhood to the Hitting Set instance $\mathcal{F}$. For the new forts, we have two objectives. First, we want the new fort neighborhood to actually provide new restrictions, i.e., it should not be already hit by the minimum hitting set $H$ of $\mathcal{F}$. This is achieved by making sure that the candidate solution $S$ is a superset of the hitting set $H$. Secondly, we want the resulting forts (i.e., the number of unobserved vertices) to be small. We achieve this heuristically by greedily considering large candidate solutions.

Specifically, we choose candidate solutions as follows. Recall, that we consider the extension problem, i.e., we have sets $X$ and $Y$ of pre-selected and excluded vertices, respectively. Moreover, let $H$ be a minimum hitting set of the current set of fort neighborhoods. Then $V$ is partitioned into the four sets $X$, $Y$, $H$, and $U = V \setminus H \setminus X \setminus Y$. Each candidate solution $S$ we consider is a superset of $H \cup X$ and a subset of $H \cup X \cup U$. We randomly order the vertices in $U = \{u_1, \dots, u_\ell\}$ and define a sequence $U_0, \dots, U_\ell \subseteq U$. We then consider the candidate solutions $S_i = H \cup X \cup U_i$ for $0 \leq i \leq \ell$. As we want to consider large candidate solutions, we start with $U_0 = U$, which clearly yields a solution as the instance would be invalid otherwise. We obtain the subset $U_i$ from $U_{i-1}$ as follows. If $S_{i-1}$ was a solution, i.e., there were no unobserved vertices, then $U_i = U_{i-1} \setminus \{u_i\}$. Otherwise, $U_i = U_{i-1} \cup \{u_{i-1}\} \setminus \{u_i\}$. Note that this makes sure that each candidate solution $S_i$ we consider is either a solution or barley not a solution as $S_i \cup \{u_i\}$ is a solution.

This gives us at least one and up to $\ell$ new fort neighborhoods. These are not directly added to the set $\mathcal{F}$. Instead, we first apply a simple local search to make sure that each fort is minimal with respect to inclusion. To this end, we iteratively re-select vertices from $U$ that had been removed before and check whether this still results in a non-empty fort.

We note that we only add sets to $\mathcal{F}$. Thus, we have to solve a sequence of increasing Hitting Set instances as a subroutine. To improve the performance of this, one can use lower bounds achieved in earlier iterations as lower bounds for later iterations (Hitting Set is monotone with respect to the addition of sets).

## 5 Experiments

The goal of this section is threefold. First, we evaluate the performance of our algorithm in comparison to two previous state-of-the-art approaches. Secondly, we give a more detailed view on the performance by analyzing how the upper and lower bounds found be the different algorithms converge to the optimal solution. Thirdly, we evaluate the impact of the different reduction rules.

**Experiment Setup.** We implemented our algorithm in C++ 20 and compiled it with clang 15.0.1 with the `-O3` optimization flag. Our source code will be made publicly available on publication. For the comparison with the previous state-of-the-art, we use the MILP formulation approach by Jovanovic and Voss [13]. In the following, we refer to this algorithm with `MILP`. The second solver by Smith and Hicks [18] and is based on the implicit hitting set approach. Unfortunately, their code is not publicly available, and the paper does not specify all implementation details. To make a fair (or rather generous) comparison, we initialized their set of forts with our fort heuristic, which, as far as we can judge, leads to better results than reported in the original publication [18]. We refer to this algorithm as `MFN` (abbreviation for *minimum fort neighborhood*). For the implicit hitting set approaches, we use an MILP formulation to solve the Hitting Set instances. All MILP instances are solved using Gurobi 9.5.2 [10].

The experiments were run on a machine running Ubuntu 22.04 with Linux 5.15. The machine has two Intel®Xeon®Gold 6144 CPUs clocked at 3.5 GHz with 8 single-thread cores and 192 GB of RAM.

We used a collection of instances shipped with pandapower [19]. We further use the Eastern, Western, Texas and US instances from the powersimdata set[2] [21] based on the US electric grids. We interpret the power grids as graphs were buses are vertices and power lines and transformers are edges. Buses without attached loads or generators yield propagating vertices. For experiments on the pandapower instances, we used a timeout of 2 h and repeated each experiment 5 times. On the powersimdata instances, we used a timeout of 10 h and only repeated the experiments using our solver. For repeated experiments, we report the median result.

**Performance Comparison.** We compare the performance of our solver to the `MILP` and `MFN` approach, each with and without preprocessing by the reduction rules. To assess the performance of our approach with reduction rules, we compute the speedup compared to the lowest run time of the previous approaches without reduction rules.

[2] `https://github.com/Breakthrough-Energy/PowerSimData`

**Table 1** Run times of different combinations of PDS solvers and reduction rules on the pandapower data set. Note that $|S|$ differs from the results reported in other literature. This is to be expected because we include non-propagating vertices from the input. Further observe that some run times are given in milli- or microseconds.

| instance | $\|S\|$ # | MILP[a)] s | MILP+R[a)] s | MFN[a)] s | MFN+R[a)] s | Ours s | Ours+R s | Speedup |
|---|---|---|---|---|---|---|---|---|
| 4gs | 2 | 1.41m | 2.75m | 2.34m | 1.89m | **513µ** | 672µ | 2.1 |
| 5 | 2 | 1.06m | 840µ | 2.24m | 1.60m | 265µ | **254µ** | 4.2 |
| 6ww | 1 | 1.19m | 10µ | 836µ | **6µ** | 156µ | 8µ | 104.6 |
| 9 | 2 | 2.61m | 1.23m | 4.98m | 2.14m | 544µ | **181µ** | 14.4 |
| 11_iwamoto | 2 | 4.59m | 23µ | 3.49m | 21µ | 613µ | **17µ** | 205.5 |
| 14 | 3 | 1.79m | 1.38m | 3.82m | 2.01m | **487µ** | 522µ | 3.4 |
| 24_ieee_rts | 6 | 4.19m | 4.87m | 5.59m | 5.50m | 1.30m | **802µ** | 5.2 |
| 30 | 6 | 2.93m | 52µ | 5.01m | 49µ | 622µ | **45µ** | 65.1 |
| ieee30 | 6 | 3.40m | 52µ | 6.53m | **43µ** | 669µ | 46µ | 73.9 |
| 33bw | 11 | 1.81m | 50µ | 8.39m | **44µ** | 551µ | 144µ | 12.6 |
| 39 | 9 | 6.60m | 112µ | 11.95m | **104µ** | 1.76m | 116µ | 56.9 |
| 57 | 12 | 9.74m | 9.78m | 24.94m | 13.83m | **1.71m** | 1.75m | 5.6 |
| 89pegase | 13 | 22.73m | 190µ | 12.88m | **169µ** | 1.74m | 288µ | 44.7 |
| 118 | 29 | 14.26m | 12.49m | 59.64m | 39.27m | 7.63m | **4.41m** | 3.2 |
| 145 | 18 | 123.65m | 275µ | 19.65m | **269µ** | 2.46m | 274µ | 71.7 |
| illinois200 | 39 | 20.57m | **276µ** | 162.35m | 464µ | 4.56m | 412µ | 49.9 |
| 300 | 72 | 29.66m | 2.75m | 218.41m | 4.70m | 7.57m | **1.44m** | 20.6 |
| 1354pegase | 311 | 105.77m | 2.61m | 2.10 | 2.93m | 19.37m | **1.89m** | 56.0 |
| 1888rte | 375 | 554.12m | 6.82m | 6.26 | 5.80m | 40.36m | **3.10m** | 178.7 |
| 2848rte | 585 | 603.93m | 7.55m | 15.77 | 6.20m | 52.60m | **4.94m** | 122.3 |
| 2869pegase | 612 | 1.21 | 221.46m | 16.12 | 1.53 | 165.41m | **96.05m** | 12.6 |
| 3120sp | 768 | 1.36 | 270.73m | 40.29 | 1.60 | 310.77m | **113.29m** | 12.0 |
| 6470rte | 1303 | 2.94 | 42.65m | 88.96 | 68.06m | 241.73m | **27.58m** | 106.7 |
| 6495rte | 1314 | 3.52 | 45.85m | 89.29 | 91.92m | 256.41m | **27.12m** | 129.7 |
| 6515rte | 1315 | 3.89 | 45.88m | 89.75 | 98.39m | 231.60m | **27.83m** | 139.9 |
| 9241pegase | 2010 | 5.71 | 1.31 | 212.93 | 13.47 | 1.36 | **660.18m** | 8.6 |

a) numbers here were obtained from our interpretation of the respective approach

Table 1 shows the run times of the solvers on the smaller pandapower instances. Preprocessing significantly reduced the running times of all solvers in most cases, especially for the larger instances. We found that our solver with reduction rules performs best in 17 out of 26 instances. In 3 further instances, our solver without reduction rules performed best and the version with reduction rules came second. In particular, our solver performs best on all instances of more than 300 vertices and all instances that took more than one millisecond to solve for any solvers. Even in the six instances where our solver was not the fastest, the other approaches could only compete when combined with the reduction rules.

For the larger powersimdata instances, neither MILP nor MFN were able to compute an optimal solution without using our reduction rules. Thus, for these instances, we only compare our solver with MFN+R and MILP+R. Table 2 shows the results. Observe that for Eastern, our algorithm finished after 16 min while MFN did not finish after more than 6 h, with a lower bound that was still more than 100 vertices below the optimal solution. Further observe that the number of fort neighborhoods $|\mathcal{F}|$ is lower for MFN. This is to be expected as minimizing their number is basically the main goal of MFN when finding new fort neighborhoods. However, this clearly does not show any benefit in the resulting run time.

**Table 2** Comparison between our algorithm and MFN on the larger powersimdata US instances preprocessed with our reduction rules. $n$ is the number of vertices, $|Z|$ is the number of non-propagating vertices an $|\mathcal{F}|$ is the size of the arising hitting set instance. For the solvers, we report the power dominating number $\gamma_P$ (or the best found lower bound) as well as the number of fort neighborhoods $\mathcal{F}$ and the run time.

| | Input | | Our Solver | | | MFN+R | | | MILP+R | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instance | $n$ | $\|Z\|$ | $\gamma_P$ | $\|\mathcal{F}\|$ | $t$ (s) | $\gamma_P$ | $\|\mathcal{F}\|$ | $t$ (s) | $\gamma_P$ | $t$ (s) |
| Texas | 2000 | 376 | 411 | 838 | 0.98 | 411 | 659 | 17.73 | 411 | 1.81 |
| Western | 10024 | 4106 | 1825 | 2618 | 1.55 | 1825 | 2010 | 158.51 | 1825 | 2.16 |
| Eastern | 70047 | 30332 | 12895 | 27019 | 552.46 | >12789 | >15043 | >10 h | >12890 | >10 h |
| USA | 82071 | 34814 | 15131 | 30357 | 728.62 | >14124 | >16391 | >10 h | >15126 | >10 h |

**Lower and Upper Bounds.** We note that all three approaches find lower bounds while solving the instances. In case of the implicit hitting set approach, each time we solve the current Hitting Set instance, the solution size is a lower bound for a minimum power dominating set. This yields lower bounds for our approach as well as for MFN. Moreover, Gurobi also provides lower bounds for MILP. Additionally, Gurobi provides upper bounds. To also get upper bounds for the implicit hitting set approaches, we use the following greedy heuristic. Whenever we have computed a hitting set $H$ of the current fort neighborhoods, we greedily add vertices to $H$, preferably selecting vertices with many unobserved neighbors, until we have a power dominating set. Afterwards, we make sure that the resulting solution is minimal with respect to inclusion.

With this, we can observe how quickly the different algorithms converge towards the optimal solution. Figure 4 illustrates the behavior of the bounds with respect to the time for two of the four powersimdata instances. All three algorithms use our reduction rules (recall that neither MILP nor MFN were able to solve these instances without them). We clearly see that, with our approach, the gap between upper and lower bounds shrinks quickly, in particular compared to MFN. This validates our assumption that adding many – potentially larger – forts instead of a single minimum size one is highly beneficial. Recall that MFN can increase its lower bound only by at most 1 after finding a new hitting set while we can increase the lower by up to one for each undecided unhit vertex.

Interestingly, for MILP+R the gap between upper and lower bound closes much quicker than for MFN+R. In particular, for the largest USA instance, there is almost no gap left after little more than 100 s. Gurobi also found an optimal solution, but failed to prove the lower bound on its size within the timeout of 10 h. Thus, in cases where a good approximation is acceptable, MILP+R is not much worse than our approach.

**Reduction Rules.** To evaluate the effect of the reduction rules on the performance of our algorithm, we let it run on the pandapower instances with different subsets of reduction rules. Recall that we have several local reduction rules as well was the two non-local rules Dom and NecN. In addition to using all or no reduction rules, we consider the following subsets. Only local rules, only non-local rules, all local rules together with Dom, and all local rules together with NecN.

Figure 5a shows the median running time for each instance in the different settings. In most instances, the reductions could decrease the running time by an order of magnitude or more. Moreover, we can see that in most cases all reduction rules are relevant, i.e., we achieve the lowest run time when using all reduction rules and applying no reduction rules is usually slower than applying any of the rules.

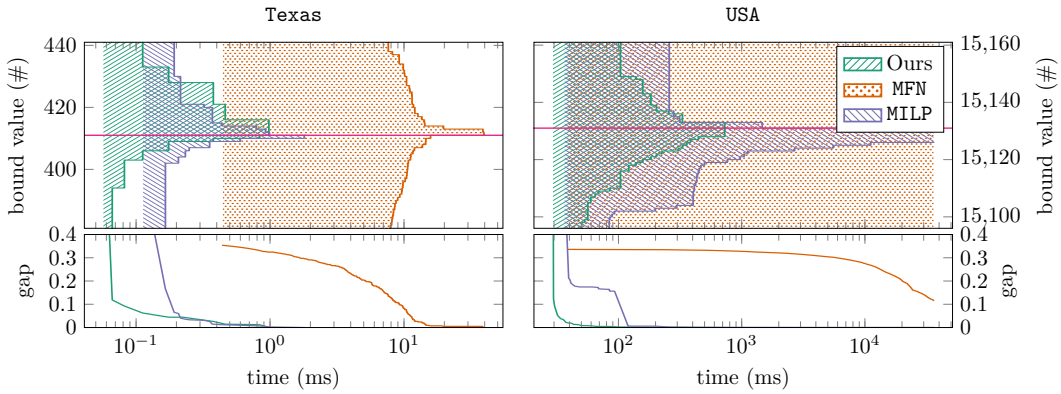



**Figure 4** Upper and lower bounds on the optimum value on the `Texas` and `USA` powersimdata instances with preprocessing by our reduction rules. We give the bounds reported by our solver and by `MFN`, both with added greedy upper bounds, as well as Gurobi for `MILP`. Lines and shaded areas each start at the time of the first respective bound. Note that the x axis uses a logarithmic scale.

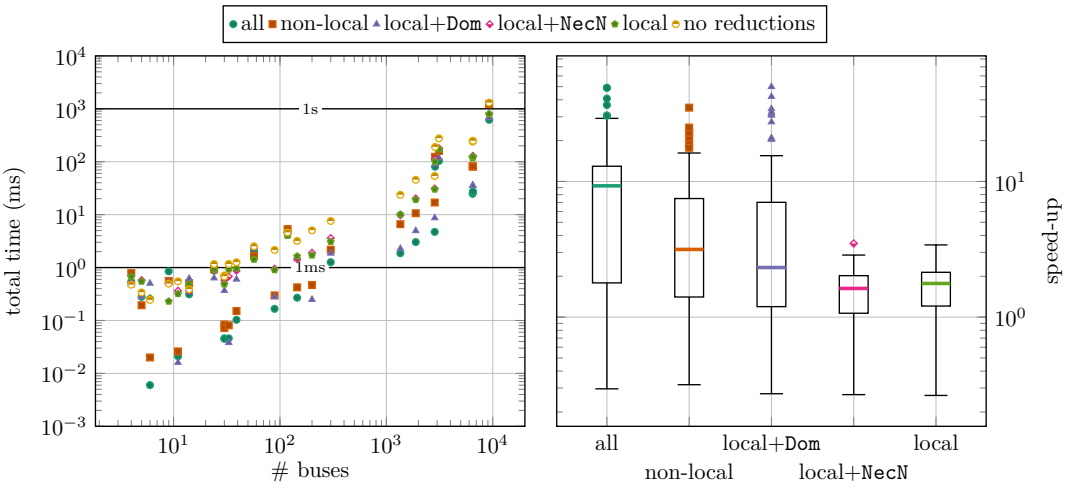
Figure 5b shows the speedup aggregated over all instances of using reduction rules compared to using no reduction rules for our solver. We can see that the median speedup is roughly one order of magnitude when applying all reduction rules. The most interesting observation here is that local+`NecN` does not give any improvement compared to just local. In fact, it is slightly slower. However, when combined with `Dom`, `NecN` gives a significant improvement.

## 6 Conclusion

We showed that PDS is $W[P]$-complete. This closes the gap in the study of its parameterized complexity. Our reduction uses an auxiliary problem, IPDS, to simulate arbitrary monotone circuits.

Our second contribution in this paper is a set of new reduction rules for PDS. The rules yield partially solved instances of PDS-Extension where some vertices are pre-selected for the power dominating set while other are forbidden from being included. Each rule shrinks the instance by removing vertices or edges, or pre-selects or excludes vertices from being selected. Our reduction rules can be used as a pre-processing step to significantly enhance the performance of existing solvers. Our third and last contribution is a new algorithm for solving PDS based on the implicit hitting set approach. The core of our algorithm is a new heuristic to find missing sets for the implicit hitting set instances. We evaluate the effectiveness of our reduction rules and the performance of our algorithm in experiments on a set of practical power grid instances from the literature. For comparison, we run the same experiments with two different approaches from the literature. The comparison shows clearly that our new heuristic for finding missing fort neighborhoods outperforms the previous approach. Our algorithm outperforms the reference solvers by more than one order of magnitude. Even when combining the other approaches with our reduction rules, our algorithm beats them on most instances. Furthermore, we can solve large instances of continental scale that could not be solved before. We found that our algorithm finds lower bounds on the power dominating number more quickly than Gurobi.

A major advantage of our fort heuristic is that it translates easily to other variants of PDS, as long as it is easy to verify which vertices are observed by a partial solution. Examples of such variant are the $k$-Power Dominating Set where propagation is possible




**(a)** Median running time on each instance with the different subsets of reduction rules.

**(b)** Aggregated speed-up of each set of reduction rules compared to our algorithm without reduction rules.

**Figure 5** Running times and speed-up of our algorithm with different subsets of the reduction rules on the pandapower instances.

if a vertex has less than $k$ unobserved neighbors or $l$-Round Power Dominating Set where the number of propagation steps is limited. Other variants, such as Connected Power Dominating Set are less straightforward. It might be interesting to see if connectivity can be efficiently enforced in the implicit hitting set model.

Even though our algorithm shows a significant improvement over the state-of-the-art, there is still some potential for further engineering. Currently, our implementation of the reduction rules is optimized for a single execution as a pre-processing step. Further optimization might make them more efficient, especially when only few vertices have changed between rule applications. This might be useful in more accurate heuristics solutions on large instances or for use in a branching algorithm. Further fast high quality heuristics can provide good upper bounds on the solution size. Such a heuristic, combined with the lower bound provided by our algorithm, might prove optimality earlier, further reducing the run time. Also, other hitting set solvers beside Gurobi exist and our algorithm might benefit from using those instead.

## References


1 Ashkan Aazami. Domination in graphs with bounded propagation: algorithms, formulations and hardness results. *Journal of Combinatorial Optimization*, 19:429–456, 2008. doi:10.1007/s10878-008-9176-7.

2 T. L. Baldwin, L. Mili, M. B. Boisen, and R. Adapa. Power system observability with minimal phasor measurement placement. *IEEE Transactions on Power Systems*, 8:707–715, 1993. doi:10.1109/59.260810.

3 Daniel Binkele-Raible and Henning Fernau. An exact exponential time algorithm for power dominating set. *Algorithmica*, 63(1):323–346, 2012. doi:10.1007/s00453-011-9533-2.

4 Thomas Bläsius and Max Göttlicher. An efficient algorithm for power dominating set, 2023. arXiv:2306.09870.

5 Chassidy Bozeman, Boris Brimkov, Craig Erickson, Daniela Ferrero, Mary Flagg, and Leslie Hogben. Restricted power domination and zero forcing problems. *Journal of Combinatorial Optimization*, 37:935–956, 2018. doi:10.1007/s10878-018-0330-6.

6 Boris Brimkov, Derek Mikesell, and Logan Smith. Connected power domination in graphs. *Journal of Combinatorial Optimization*, 38(1):292–315, 2019. doi:10.1007/s10878-019-00380-7.

7 Dennis J. Brueni. Minimal pmu placement for graph observability: a decomposition approach. Master's thesis, Virginia Polytechnic Institute and State University, 1993. doi:10919/45368.

8 Dennis J. Brueni and Lenwood S. Heath. The pmu placement problem. *SIAM J. Discret. Math.*, 19:744–761, 2005. doi:10.1137/S0895480103432556.

9 Jiong Guo, Rolf Niedermeier, and Daniel Raible. Improved algorithms and complexity results for power domination in graphs. *Algorithmica*, 52(2):177–202, 2008. doi:10.1007/s00453-007-9147-x.

10 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

11 Teresa W Haynes, Sandra M Hedetniemi, Stephen T Hedetniemi, and Michael A Henning. Domination in graphs applied to electric power networks. *SIAM journal on discrete mathematics*, 15(4):519–529, 2002. doi:10.1137/S0895480100375831.

12 Mikoláš Janota and Joao Marques-Silva. Solving qbf by clause selection. In *International Joint Conference on Artificial Intelligence*, 2015.

13 Raka Jovanovic and Stefan Voss. The fixed set search applied to the power dominating set problem. *Expert Systems*, 37(6):e12559, 2020. doi:10.1111/exsy.12559.

14 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Parameterized power domination complexity. *Information Processing Letters*, 98(4):145–149, 2006. doi:10.1016/j.ipl.2006.01.007.

15 Chung-Shou Liao and Der-Tsai Lee. Power domination in circular-arc graphs. *Algorithmica*, 65(2):443–466, 2013. doi:10.1007/s00453-011-9599-x.

16 L. Mili, Thomas L. Baldwin, and R. Adapa. Phasor measurement placement for voltage stability analysis of power systems. *29th IEEE Conference on Decision and Control*, pages 3033–3038 vol.6, 1990. doi:10.1109/CDC.1990.203341.

17 Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP Hybrid MaxSAT Solver. In *International Conference on Theory and Applications of Satisfiability Testing*, 2016. doi:10.1007/978-3-319-40970-2_34.

18 Logan A. Smith and Illya V. Hicks. Optimal sensor placement in power grids: Power domination, set covering, and the neighborhoods of zero forcing forts, 2020. arXiv:2006.03460.

19 L. Thurner, A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun. pandapower – An open-source python tool for convenient modeling, analysis, and optimization of electric power systems. *IEEE Transactions on Power Systems*, 33(6):6510–6521, November 2018. doi:10.1109/TPWRS.2018.2829021.

20 Guangjun Xu, Liying Kang, Erfang Shan, and Min Zhao. Power domination in block graphs. *Theoretical computer science*, 359(1-3):299–305, 2006. doi:10.1016/j.tcs.2006.04.011.

21 Yixing Xu, Nathan P Myhrvold, Dhileep Sivam, Kaspar Mueller, Daniel Julius Olsen, Bainan Xia, Daniel Livengood, Victoria Hunt, Benjamin Rouill'e d'Orfeuil, Daniel B. C. Muldrew, Merrielle Ondreicka, and Megan Bettilyon. U.s. test system with high spatial and temporal resolution for renewable integration studies. *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5, 2020. doi:10.1109/PESGM41954.2020.9281850.

# Incremental $(1-\varepsilon)$-Approximate Dynamic Matching in $O(\text{poly}(1/\varepsilon))$ Update Time

**Joakim Blikstad**
KTH Royal Institute of Technology, Stockholm, Sweden

**Peter Kiss**
Max-Planck-Institut für Informatik, Saarbrücken, Germany
University of Warwick, Coventry, UK

## Abstract

In the dynamic approximate maximum bipartite matching problem we are given bipartite graph $G$ undergoing updates and our goal is to maintain a matching of $G$ which is large compared the maximum matching size $\mu(G)$. We define a dynamic matching algorithm to be $\alpha$ (respectively $(\alpha, \beta)$)-approximate if it maintains matching $M$ such that at all times $|M| \geq \mu(G) \cdot \alpha$ (respectively $|M| \geq \mu(G) \cdot \alpha - \beta$).

We present the first deterministic $(1-\varepsilon)$-approximate dynamic matching algorithm with $O(\text{poly}(\varepsilon^{-1}))$ amortized update time for graphs undergoing edge insertions. Previous solutions either required super-constant [Gupta FSTTCS'14, Bhattacharya-Kiss-Saranurak SODA'23] or exponential in $1/\varepsilon$ [Grandoni-Leonardi-Sankowski-Schwiegelshohn-Solomon SODA'19] update time. Our implementation is arguably simpler than the mentioned algorithms and its description is self contained. Moreover, we show that if we allow for additive $(1, \varepsilon \cdot n)$-approximation our algorithm seamlessly extends to also handle vertex deletions, on top of edge insertions. This makes our algorithm one of the few small update time algorithms for $(1-\varepsilon)$-approximate dynamic matching allowing for updates both increasing and decreasing the maximum matching size of $G$ in a fully dynamic manner.

Our algorithm relies on the weighted variant of the celebrated Edge-Degree-Constrained-Subgraph (EDCS) datastructure introduced by [Bernstein-Stein ICALP'15]. As far as we are aware we introduce the first application of the weighted-EDCS for arbitrarily dense graphs. We also present a significantly simplified proof for the approximation ratio of weighed-EDCS as a matching sparsifier compared to [Bernstein-Stein], as well as simple descriptions of a fractional matching and fractional vertex cover defined on top of the EDCS. Considering the wide range of applications EDCS has found in settings such as streaming, sub-linear, stochastic and more we hope our techniques will be of independent research interest outside of the dynamic setting.

**2012 ACM Subject Classification** Theory of computation → Dynamic graph algorithms; Theory of computation → Approximation algorithms analysis

**Keywords and phrases** Bipartite Matching, Incremental Matching, Dynamic Algorithms, Approximation Algorithms, EDCS

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.22

**Related Version** *Full Version*: `https://arxiv.org/abs/2302.08432` [27]

**Funding** *Joakim Blikstad*: Partially supported by the Swedish Research Council (Reg. No. 2019-05622).
*Peter Kiss*: Partially supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S03353X/1.

**Acknowledgements** We thank Danupon Nanongkai for insightful discussions and valuable comments throughout the development of this work. Part of this work was done while the authors visited the Max Planck Institute of Informatics, Saarbrücken.




# 1 Introduction

Matchings are fundamental objects of combinatorical optimization with a wide range of practical applications. The first polynomial time algorithm for finding a maximum matching was published by Kuhn[1] [39] in 1955 which ran in $O(n^4)$ time. A long line of papers have focused on improving this polynomial complexity. Notably Edmonds and Karp [30] showed the first $O(n^3)$ time algorithm for the problem which was later improved to $O(n^{2.5})$ [36]. Mucha and Sankowski [40] showed maximum matching can be solved in matrix multiplication time, that is in $O(n^\omega)$ where $\omega$ is currently around 2.38. In the very recent breakthrough result of Chen-Kyng-Liu-Peng-ProbstGutenberg-Sachdeva [29], they showed an $O(m^{1+o(1)})$ time algorithm for the maximum flow problem (which generalizes bipartite matching) providing the first near-linear time algorithm, essentially settling the problem in the sequential setting.

**Dynamic Model.** This paper focuses on the matching problem in the dynamic model where it has received extensive research attention in recent years, see e.g [1, 2, 10, 14, 17, 20, 22, 28, 41, 43, 45, 47] and many more. In this setting our task is to maintain a large matching as the graph undergoes updates. We will refer to updates being fully dynamic if they concern both insertions and deletions and partially dynamic if only one of the two is allowed. The objective is to minimize the time spent maintaining the output after each update. Throughout the paper we will always refer to update time in the amortized sense – averaged over the sequence of updates. In [45] Sankowski has shown the fist improvement for the fully dynamic maximum matching problem in terms of update time ($O(n^{1.45})$) compared to static recomputation after each update. Unfortunately, based on well accepted hardness conjectures no dynamic algorithm for the maximum matching problem may achieve update time sub-linear in $n$ [35]. Most works focused on the relaxed approximate version of the problem where the goal is to maintain a large matching in $G$ with respect to the maximum matching size $\mu(G)$. We will refer to matching algorithms as $\alpha$-approximate (respectively $(\alpha, \beta)$-approximate) if it maintains matching $M$ such that at all times $|M| \geq \mu(G) \cdot \alpha$ (respectively $|M| \geq \mu(G) \cdot \alpha - \beta$).

**Fully Dynamic Approximate Matching.** The holy grail of dynamic algorithm design is to achieve an update time of $O(\text{polylog}(n))$ or ideally even constant. For the fully dynamic approximate matching problem, a long line of research [6, 9, 18, 19, 21, 26, 42, 46] has lead to algorithms achieving $\approx \frac{1}{2}$-approximation with $O(\text{polylog}(n))$ and constant update time. No fully dynamic algorithm has been found achieving better than $\frac{1}{2}$-approximation in $O(\text{polylog}(n))$ update time for the problem, and this challenge appears to be one of the most celebrated problem within the dynamic matching literature. A set of interesting papers focused on $\approx \frac{2}{3}$-approximation in $\tilde{O}(\sqrt{n})$ update time [16, 17, 32, 38] and other approximation-ratio to polynomial-update-time trade-offs in the better-than-$\frac{1}{2}$-approximation regime were achieved by [10, 11, 44]. Note that through periodic recomputation of the matching (roughly every $\varepsilon\mu(G)$ updates) we can achieve fully dynamic $(1-\epsilon)$-approximation in $\tilde{O}(n)$ update time [34]. Very recently [12] has shown that $(1-\varepsilon)$-approximation is possible in slightly sublinear update time $O(n/\log^*(n)^{O(1)})$ suggesting that there might exist efficient non-trivial algorithms for the problem. Note that very recently [8, 25] have independently shown that if our goal is to maintain the *size* of the maximum matching (and not the edge-set) then sub-$\frac{1}{2}$ approximation is achievable in polylogarithmic update time.

[1] However, this result is usually attributed to Kőnig and Egerváry.

**Partially Dynamic Matching Algorithms.** For small approximation ratios, achieving poly-logarithmic update time for fully dynamic matching seems far out of reach with current techniques, or perhaps even impossible. Hence, a long line of papers have focused on maintaining a $(1-\varepsilon)$-approximate matching in partially dynamic graphs: either incremental (edge insertions) or decremental (edge deletions). The first $O(\text{poly}(\log(n), \varepsilon^{-1}))$ algorithm for maintaining a $(1-\varepsilon)$-approximate matching under edge insertions is due to Gupta [33], with amortized update time $O(\log^2(n)/\varepsilon^4)$. Their algorithm models the bipartite matching problem as a linear program, and uses the celebrated multiplicative-weights-updates framework. Generalizing the result of [33] recently Bhattacharya-Kiss-Saranurak [23] has shown that an arbitrarily close approximation to the solution of a linear program undergoing updates either relaxing or restricting (but not both) its solution polytope can be maintained in $O(\text{poly}(\log n, \varepsilon^{-1}))$ update time. Hence, the algorithm of [23] shows how to maintain a $(1-\varepsilon)$-approximate matching under either decremental or incremental updates with a unified approach. As both of these papers rely on static linear program solver sub-routines their update times inherently carry $\log(n)$ factors, and it seems implausible that these techniques can achieve constant update time independent of $n$.

The decremental algorithms of [15,37] focus on maintaining "evenly spread out" fractional matchings so that they are robust against edge-deletions. These algorithms rely on either maximum-flow computation or convex optimization sub-routines which similarly to LP-solvers carry $\log(n)$-factors into the update time.

The first constant time[2] partially dynamic matching algorithm is due to [31] who solve $(1-\varepsilon)$-approximate matching in incremental graphs with an update time of $(1/\varepsilon)^{O(1/\varepsilon)}$. Their solution relies on augmenting path elimination, a technique used commonly for the matching problem in the static setting. However, enumerating augmenting paths seems to inherently carry an exponential dependency on $1/\varepsilon$ due to the number of possible such paths present in the graph.

As far as we are aware partially dynamic $(1-\epsilon)$-approximate matching algorithms with update time independent of $n$ and faster than some exponential in $\epsilon$ are all restricted to a relaxed version of the problem where the graph may undergo vertex insertions/deletions. Such an algorithm can simply be obtained through straightforward periodic rebuilds (if we allow for additive $\epsilon \cdot n$ slack) or as shown by Zheng-Henzinger [48][3]. Hence, it we can observe the following apparent gap in the literature of partially dynamic matching algorithms:

▶ **Question 1.** *Can we maintain a $(1-\varepsilon)$-approximate maximum matching of a partially dynamic bipartite graph in $O(\text{poly}(\varepsilon^{-1}))$ update time?*

Based on the current landscape of the dynamic algorithms literature, achieving $(1-\varepsilon)$-approximation under fully dynamic updates in small update times seems to be far out of reach. Contrary to the extensive research effort, no fully dynamic algorithm with $\text{poly}(\log(n), \varepsilon^{-1})$ has been found for maintaining matchings with better than $\frac{1}{2}$-approximation. This apparent difficulty proposes the research of dynamic models somewhere between fully and partially dynamic updates. The previously mentioned paper by Zheng-Henzinger [48] implements a

[2] That is constant-time whenever $\varepsilon$ is constant, i.e. the update time should be independent of $n$.

[3] A very recent paper of Zheng-Henzinger [48] has initially claimed an algorithm which can maintain a $(1-\epsilon)$-approximate matching in $O(1/\varepsilon)$ update time under edge deletions. However, the authors have found a mistake in their paper and claim that their algorithm only works under specific vertex updates

$(1-\epsilon)$-approximate algorithm with $O(1/\varepsilon)$-update time which can support vertex insertions and deletions on separate sides of the bipartition. The existence of this new result proposes the following natural question:

▶ **Question 2.** *Under what kind of non-partially dynamic updates can we maintain a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph?*

## 1.1 Our Contribution

In this paper we provide a positive answer to **Question 1** and make progress towards understanding **Question 2**. Our main result is the first $O(\text{poly}(1/\varepsilon))$ update time $(1-\varepsilon)$-approximate dynamic matching algorithm for bipartite graphs undergoing edge insertions:

▶ **Theorem 1.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph undergoing edge insertions in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

Previous algorithms for $(1-\varepsilon)$ approximate dynamic matching under edge updates required update times which were either super-constant [23, 33] or had an exponential dependency on $\varepsilon^{-1}$ [31]. Furthermore, our algorithm is arguably simpler then previous implementations and it is self contained (except for the static computation of $(1-\varepsilon)$-approximate maximum matchings) where as most dynamic matching algorithms either rely on heavy machinery from previous papers or use black-box tools like multiplicative weight updates or flow-subroutines.

We further show that if we allow for (additive) $(1, \varepsilon \cdot n)$-approximation[4] our algorithm seamlessly extends to a wider range of updates:

▶ **Theorem 2.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1, \epsilon \cdot n)$-approximate maximum matching of a bipartite graph undergoing edge insertions and vertex deletions in total update time of $O(n/\varepsilon^8 + m/\varepsilon^5)$.*

In contrast to the similar update time result of [48] which allows for edge deletions and vertex insertions on one side of the bipartition our algorithm allows from arbitrary vertex deletions. Our algorithm maintains a $(1-\varepsilon)$-approximate maximum matching of the graph throughout updates which can both increase and decrease the maximum matching size of the graph. Hence, we hope our techniques provide useful insight towards fully dynamizing $(1-\varepsilon)$-approximate algorithms for the matching problem.

Our algorithm relies on the weighted variant of the celebrated Edge-Degree-Constrained-Subgraph (EDCS) matching sparsifier. The unweighted EDCS (first introduced by Bernstein-Stein [16]) has found applications in a number of different computational settings: streaming [3, 4, 13], stochastic, one way communication, fault tolerant [5], sub-linear [8, 12, 24, 25] and dynamic [8, 16, 17, 32, 38]. On the other hand the *weighted* EDCS variant which provides a tighter approximation has only found applications in small arboricity graphs [16]. Hence, we initiate the study of the weighted EDCS in dense graphs.

Furthermore, we show a significantly simplified proof for the approximation ratio of the weighted EDCS datastructure with respect to the maximum matching size. In our proof, we identify simple and explicit descriptions of a fractional matching and fractional vertex covers

[4] Recall that this means that we maintain a matching of size at least $\mu - \varepsilon n$, as opposed to $\mu - \varepsilon\mu$, where $\mu$ denotes the size of the maximum matching.

defined on top of the weighted EDCS, which might be of independent interest. Moreover, we show that the dependence on the slack parameter on the maximum degree of the weighted EDCS is exactly quadratic. This in sharp contrast to the unweighted EDCS where the same relationship have been proven to be linear [7]. While within the dynamic matching algorithm literature papers don't tend to focus on exact $\epsilon$ complexities but rather $n$ dependence, in models such as semi-streaming and distributed the $\epsilon$ dependency usually gains more focus. Our hard example (most likely) rules out applications of weighted EDCS in these models for obtaining sub $\epsilon^{-2}$ round/pass complexity algorithms. We hope that these observations will be of independent research interest due to the wide-spread popularity of the EDCS datastructure for solving matching problems.

## 1.2 Our Techniques

Assume $H$ is a multi-graph defined on the vertex set of $G$ and let $\deg_H(v)$ stand for the degree of vertex $v$ in $H$. We define the degree of an edge to be the sum of the degrees of its endpoints.[5]

▶ **Definition 3** (Weighted EDCS [16]). *Given a graph $G = (V, E)$, a multiset $H$ is called a weighted EDCS with parameter $\beta$ if*[a]*:*

**(i)** $\deg_H(u) + \deg_H(v) \leq \beta$ *for all edges* $(u, v) \in H$.

**(ii)** $\deg_H(u) + \deg_H(v) \geq \beta - 1$ *for all edges* $(u, v) \in E$.

If $H$ is not a weighted EDCS, we call an edge $e \in H$ *overfull* if it violates (i), and an edge $e \in E$ *underfull* if it violates (ii).

[a] Some authors use an additional parameter $\beta^- < \beta$ which replaces the "$\beta - 1$" in the degree-constraint. For our purposes, we will always have $\beta^- = \beta - 1$.

If $\beta = \Omega(\varepsilon^{-2})$ and $H$ is a $\beta$-WEDCS of $G$ then $\mu(H) \geq \mu(G) \cdot (1-\varepsilon)$ ( [16], Theorem 13). In order to derive our incremental result we show that a $\beta$-WEDCS can be efficiently maintained greedily under edge insertions. In turn we can efficiently maintain a $(1 - \epsilon)$-approximate matching within the support of $H$ through periodic recomputation.

Define a valid update of $H$ to be one of the following: (i) and edge $e \in H$ which is overfull with respect to $H$ gets deleted from $H$; and (ii) a copy of an edge $e \in E$ which is underfull with respect to $H$ is added to $H$. In Lemma 7 (slightly improving on the similar lemma's of [5, 13, 16]) we show that if $H$ is initialized as the empty graph and only undergoes valid updates, then it there are at most $O(\mu(G) \cdot \beta^2)$ many updates.

Fix some $\beta = \Theta(\varepsilon^{-2})$. Assume $G$ is initially empty and initialize $H$ to be an empty edge set (note that by definition initially $H$ is a $\beta$-WEDCS of $G$). Assume edge $e$ is inserted into $G$. If at this point $e$ is not underfull with respect to $H$ there is nothing to be done as $H$ remained a valid WEDCS of $G$. If $e$ is underfull with respect to $H$ we add copies of it to $H$ until it is not. This process of adding $e$ to $H$ has increased the edge degree of edges neighbouring $e$ in $H$ and some of them might have became overfull. To counteract this we iterate through the neighbours of $e$ in an arbitrary order and if we find an overfull edge $e'$ we remove it from $H$. This edge removal decreases the edge degrees in the neighbourhood of $e'$. To counteract this we recurse and look for underfull edges in the neighbourhood of

[5] Note that we are sticking to the notation weighted-EDCS instead of multi-EDCS to be in line with the naming convention of [16] which defined $H$ to be a weighted graph with integer edge weights.

$e'$. If such an edge $e''$ is found we add copies of it to $H$ until $e''$ is not underfull and repeat the same steps as if $e''$ was just inserted into $G$. This defines a natural recursive process for restoring the WEDCS properties after each edge insertion in a local and greedy way.

Whenever we have to explore the neighbourhood of an edge in $O(\Delta)$ time (where $\Delta$ is the max-degree) to either check for underfull or overfull edges we do so because $H$ underwent a valid update. By Lemma 7 this may only happen at most $O(\mu(G) \cdot \beta^2)$ times. Hence, naively the total work spent greedily fixing the WEDCS properties is $O(\mu(G) \cdot \Delta \cdot \beta^2)$. For some graphs this value might be significantly larger then $m$. In order to improve the update time to $O_\beta(m)$ we assign a counter $c_v$ to each vertex $v$ measuring the number of valid updates of $H$ the neighbourhood of $v$ has underwent. Once $c_v$ grows to $\Omega(\beta^2 \cdot \varepsilon^{-1})$ we mark $v$ dirty and ignore further edges inserted in the neighbourhood of $v$. By marking a single vertex dirty and ignoring some edges incident on it we may loose out only on a single edge of any maximum matching. However, whenever we mark a vertex dirty we can charge $\Omega(\beta^2 \cdot \varepsilon^{-1})$ valid updates of $H$ to that vertex. As there may be at most $O(\mu(G) \cdot \beta^2)$ valid updates of $H$ in total we may only mark $O(\mu(G) \cdot \varepsilon)$ vertices dirty hence we will only ignore an $O(\epsilon)$-fraction of any maximum matching within the graph through ignoring edges incident on dirty vertices. As we may scan the neighbourhood of vertex $v$ at most $O(\beta^2 \cdot \varepsilon^{-1}) = \text{poly}(\varepsilon^{-1})$ times until $v$ is marked dirty we ensure that each edge is explored $\text{poly}(\varepsilon^{-1})$ times guaranteeing a total running time of $O(m \cdot \text{poly}(\varepsilon^{-1}))$. Full details can be found in Section 3.

**Towards Full Dynamization.** The algorithm almost seamlessly adopts to vertex deletions if we allow for $(1, \varepsilon \cdot n)$-approximation[6]. Whenever a vertex gets deleted from the graph our WEDCS $H$ might be locally affected. This means that over the full run of the algorithm, $H$ may undergo further valid updates then the $O(\mu(G) \cdot \beta^2)$ bound provided by Lemma 7. A potential function based argument allows us to claim that each vertex deletion may increase the total number valid updates $H$ may undergo by $O(\beta^2)$. As each vertex may be deleted at most once this means that the total number of valid updates we might make to restore $H$ is $O(n \cdot \beta^2)$, each update requiring $O(\Delta)$ time if naively implemented. By marking vertices as dirty as before we can guarantee amortized $O(\text{poly}(1/\varepsilon))$ update time. However, now we must mark up to $\approx \varepsilon \cdot n$ vertices as dirty (as opposed to $\approx \varepsilon \cdot \mu(G)$ like before), which means we may miss out on $\varepsilon \cdot n$ edges of the maximum matching.

## 2 Preliminaries

**Matching Notation.** Let $N_E(v)$ stand for the edges neighbouring vertex $v$ in $E$. A fractional matching $f$ of a graph $G$ is an assignment of the edges of $G$ to values in the range $[0, 1]$ such that for all vertices $v \in V$ it holds that $\sum_{e \in N_E(v)} f_e \leq 1$. The *size* of a fractional matching is simply the sum of the fractional values over its edges. That is a maximum fractional matching is the solution to the linear program $\max\{\sum_{e \in E} f_e : \sum_{e \in N_E(v)} f_e \leq 1 \text{ for all } v \in V, f \geq 0\}$. A solution $x$ to the dual of this program $\min\{\sum_{v \in V} x_v : x_u + x_v \geq 1 \text{ for all } (u, v) \in E, x \geq 0\}$ is a fractional vertex cover.

[6] Readers may reasonable argue that the additive slack is not necessary as a number of vertex-sparsification techniques exist in literature allowing us to improve the approximation to purely multiplicative slack in the dynamic setting. Unfortunately, these techniques don't appear to be robust against vertex-wise updates.

Approximation with respect to a fractional matching is defined simmilarly as with respect to integral matchings. For a graph $G = (V, E)$ we use $\mu(G)$ to denote the size of the maximum matching in $G$. Likewise, we use $\mu^*(G)$ to denote the size of the maximum *fractional* matching. It is well-known that $\mu(G) \leq \mu^*(G) \leq \frac{3}{2}\mu(G)$ for any graph, and that $\mu^*(G) = \mu(G)$ in bipartite graphs.

▶ **Theorem 4** (Hopcroft-Karp [36]). *There exists a deterministic static algorithm which finds a $(1-\varepsilon)$-approximate maximum matching of a graph $G$ on $m$ edges in $O(m/\varepsilon)$ time.*

# 3 Incremental Approximate Matching

We start by showing our incremental fractional matching algorithm, and then show how to extend it (for bipartite graphs) to also maintain an integral matching.

## 3.1 Weighted EDCS & Fractional Matchings

In this section we show our algorithm to (almost[7]) maintain a weighted EDCS $H$ in an incremental graph. It is well-known that such an $H$ will be a $(1-\varepsilon)$-matching sparsifier on *bipartite* graphs, that is a "sparse" subgraph with $\mu(H) \geq (1-\varepsilon)\mu(G)$ [16].

As we show later in Section 5.1 (Theorem 13), we even known something stronger: there is an explicit fractional matching in $H$ of size at least $(1-\varepsilon)\mu^*(G)$, defined as

$$f_{(u,v)} = \min\left(\frac{1}{\deg_H(v)}, \frac{1}{\deg_H(u)}\right) \text{ on each } (u,v) \in H. \tag{1}$$

Note that [17] similarly (implicitly) defines a large fractional matching on the support of a weighted EDCS, however our construction and analysis are arguably simpler. This fractional matching is also valid for general (non-bipartite) graphs. Hence our incremental algorithm will also maintain this explicit $(1-\varepsilon)$-approximate fractional matching (even in non-bipartite graphs). Formally we prove the following theorem.

▶ **Theorem 5.** *For any $\varepsilon \in (0,1)$, there is an algorithm (Algorithm 1) that maintains a $(1-\varepsilon)$-approximate maximum* fractional *matching in an incremental graph in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$. Additionally, this fractional matching is always supported on a set of edges $H$ of size $|H| \leq \Theta(\mu(G)/\varepsilon^2)$ and maximum degree $O(1/\varepsilon^2)$.*

First we need two standard facts about weighted EDCS. For completeness, we prove these in Section 7. Lemma 7 has only been shown before for unweighted EDCS [5, 13, 16] and not weighted (but the arguments are very similar).

▶ **Lemma 6.** *In a $\beta$-WEDCS $H$, the maximum degree is at most $\beta$ and $|H| \leq \beta\mu^*(G)$.*

▶ **Lemma 7.** *If a multiset of edges $H$ is only ever changed by removing overfull edges and adding underfull edges, then there are at most $\beta^2\mu^*(G)$ such insertions/deletions to $H$.*

[7] As we will see later in this section, our sparsifier $H$ will be a weighted EDCS for $G \setminus R$, where $R$ is a subgraph of $G$ with very small maximum matching size $\mu(R) = O(\varepsilon\mu(G))$.

**Overview.** Our algorithm (see Algorithm 1) will maintain a weighted EDCS $H$ with $\beta = \Theta(1/\varepsilon^2)$. We also maintain the $(1-\varepsilon)$-approximate fractional matching $f$ as in Equation (1) and Theorem 13.

When we get an edge-insertions $(u, v)$, we need to reestablish the property that $H$ is an EDCS. If $(u, v)$ is underfull ($\deg_H(u) + \deg_H(v) < \beta - 1$), we add it (maybe multiple times) to $H$. This means that $\deg_H(u)$ (similarly $\deg_H(v)$) increases, which can potentially make some incident edge $(u, w) \in H$ overfull ($\deg_H(u) + \deg_H(w) > \beta$), so we must remove one such edge. This might in turn lead to some edge $(w, z) \in E$ being underfull (as now $\deg_H(w)$ decreased), so we add this edge to $H$. This process continues, so both from $u$ and $v$ we need to search for alternating paths of underfull and overfull edges (as is standard in EDCS-based algorithms). In total, Lemma 7 says there are $O(n/\varepsilon^4)$ updates to $H$ over the full run of the algorithm.

We note that searching for an *overfull* edge is cheap: the maximum degree in $H$ is just $O(\beta)$ (Lemma 6), so we can afford to, in $\Theta(1/\varepsilon^2)$ time, check all incident edges. However, searching for *underfull* edges is more expensive: this time we cannot afford to just go through all neighboring edges in $E$, as we no longer have a bound on the maximum degree.

To overcome this we use an amortization trick which allows us to ignore a vertex if we touched it too many times. There are only $\beta^2\mu^*(G)$ updates to $H$ in total (Lemma 7), so there will only be $\varepsilon\mu^*(G)$ many vertices incident to more than $2\beta^2/\varepsilon$ of these updates. Any edges incident to these "update-heavy" vertices we may ignore, as this may only decrease the maximum matching size by an $\varepsilon$ fraction. We thus only need to check each edge a total of $O(1/\varepsilon^5)$ times over the run of the algorithm, except when it is in $H$ already. Note that $H$ is no longer a weighted EDCS of $G = (V, E)$, but rather of $G' = (V, (E \setminus R) \cup H)$ where $R$ is this set of edges we ignored (with $\mu^*(R) \leq \varepsilon\mu^*(G)$).

**Algorithm 1** Incremental Weighted EDCS & Frasctional Matching.

```
// Initially E = H = ∅ and deg_H(v) = visits[v] = 0 for all v ∈ V.
// When an edge insertion e appears, add it to E and call FixEdge(e).

function FixEdge(e = (u, v))
    if deg_H(u) + deg_H(v) > β and (u, v) ∈ H then                  // overfull
        Remove (one copy of) the edge (u, v) from H
    if deg_H(u) + deg_H(v) < β − 1 then                             // underfull
        Add (one copy of) the edge (u, v) to H
    if the edge was added or removed then
        Update deg_H(u), deg_H(v), and the fractional matching accordingly
        FixVertex(u), FixVertex(v)

function FixVertex(v)
    visits[v] ← visits[v] + 1
    if visits[v] < 2β²/ε then
        for edge e ∈ E incident to v do
            FixEdge(e)
    else
        for edge e ∈ H incident to v do
            FixEdge(e)
```

**Running Time.** We analyse the total update time spent in different parts of our algorithm.

- We first note that `FixEdge` runs in constant time whenever it does not update $H$. It is called once for each edge-insertion (total $O(m)$ times), and also some number of times from `FixVertex`.
- Now consider the case when `FixEdge` does update $H$ (which happens at most $\beta^2\mu^*(G)$ times per Lemma 7). Now the algorithm uses $O(\beta)$ time for the update of the fractional matching and insertion/removal in $H$, in addition to exactly two calls to `FixVertex`. Except for these calls to `FixVertex`, over the run of the algorithm we hence spend a total of $O(\mu(G)\beta^3) = O(n/\varepsilon^6)$ time.
- By the previous point, we will call `FixVertex` at most $2\beta^2\mu^*(G)$ times. In each call, we either loop through all incident edges in $H$ or $E$. If we loop through $H$, we visit $\beta$ many edges (by Lemma 6). Either these `FixEdge` calls take constant time, or they are already accounted for in the previous point. In total, this thus accounts for another $O(\mu(G)\beta^3) = O(n/\varepsilon^6)$ running time.
- We account for the case when `FixVertex` loops through all incident edges in $E$ differently. Consider how often a specific edge $e$ appears in the for-loop at line 13. Each endpoint vertex of $e$ will reach this line at most $O(\beta^2/\varepsilon)$ times. Hence, in total for all edges, line 13 is run at most $O(m\beta^2/\varepsilon) = O(m/\varepsilon^5)$ times.

**Approximation Guarantee.** We now argue the approximation ratio. We will show that the fractional matching supported on $H$ is a $(1-2\varepsilon)$-approximation of maximum fractional matching in $G$. If one want a $(1-\varepsilon')$-approximation, then one can run the algorithm in the same asymptotic update time setting $\varepsilon = \varepsilon'/2$, and changing $\beta$ accordingly.

Define $R_V$ to be the set of "dirty"/"update-heavy" vertices: that is vertices $v$ for which `FixVertex(`$v$`)` has been called at least $2\beta^2/\varepsilon$ many times (i.e. $\mathsf{visits}[v] \geq 2\beta^2/\varepsilon$). By a counting argument $|R_V| \leq 2\beta^2\mu^*(G)/(2\beta^2/\varepsilon) = \varepsilon\mu^*(G)$, since by Lemma 7 in total there are only $\beta^2\mu^*(G)$ many updates to $H$, each issuing exactly two calls to `FixVertex`. If $R_E$ is the set of edges incident to $R_V$, then $\mu^*(R_E) \leq |R_V| \leq \varepsilon\mu^*(G)$ since $R_V$ is a vertex cover of $R_E$.

Define $G' = (V, E \setminus (R_E \setminus H))$. By the above, $\mu^*(G') \geq (1-\varepsilon)\mu^*(G)$. We will finish the proof by arguing that $H$ is a weighted EDCS of $G'$, and thus that our fractional matching is of value at least $(1-\varepsilon)\mu^*(G') \geq (1-\varepsilon)^2\mu^*(G) \geq (1-2\varepsilon)\mu^*(G)$. Whenever an edge is added or removed to $H$, we call `FixVertex` on its endpoints, and no other degrees $\deg_H$ have changes. Every time `FixVertex`$(v)$ is called for $v \notin R_V$, we make sure that all edges $e \in E$ incident to it satisfy the definition of an EDCS, and when `FixVertex(`$v$`)` is called for some $v \in R_v$, we check the edges incident to $H$. We note that when a vertex becomes "update-heavy" (added to $R_v$), then we do **not** immediately remove all incident edges from $H$ (as then we no longer have the same bound on the number of updates to $H$ since Lemma 7 no longer applies).

▶ Remark 8. We note that our algorithm runs in time $O(n\beta^3 + m\beta^2/\varepsilon)$, and a valid question is whether setting $\beta = \Theta(1/\varepsilon^2)$ is actually necessary? Recently it was shown that for unweighted EDCS $\beta = \Theta(1/\varepsilon)$ is enough [7]. However, for weighted EDCS the $\varepsilon^2$ dependency is indeed necessary, as we show by an example where this is asymptotically tight in Section 5.2 (Theorem 16).

## 3.2 Integral Matchings in Bipartite Graphs

In this section we argue how to extend our fractional matching algorithm (Theorem 5) to maintain an integral matching instead (for bipartite graphs), in the same asymptotic update time. We cannot use known dynamic rounding techniques [23, 47], since all these incur

polylog$(n)$ factors or require randomization, and we are aiming for update time independent of $n$. In fact, our technique is simple and combinatorial; and only relies on the standard Hopcroft-Karp algorithm for finding a $(1-\varepsilon)$-approximate matching in the static setting [36].

▶ **Theorem 1.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph undergoing edge insertions in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

▶ Remark 9. Before proving the above theorem, we briefly explain how to achieve a slightly less efficient version (amortized $O(1/\varepsilon^8)$ update time) by using the fully dynamic $(1-\varepsilon)$-approximate matching algorithm of Gupta-Peng [34] as a black box. The idea is to run the fully dynamic algorithm on our sparsifier – the weighted EDCS $H$. This way we maintain a matching $M$ of size $|M| \geq (1-\varepsilon)\mu(H) \geq (1-\varepsilon)^2\mu(G) \geq (1-2\varepsilon)\mu(G)$.

Gupta-Peng [34] state that their algorithm runs in $O(\sqrt{m}/\varepsilon^2)$ time per update. However, as previously pointed out by e.g. [16, Lemma 1], it is in fact more efficient when the max-degree $\Delta$ is low, in which case the update time is only $O(\Delta/\varepsilon^2)$. Since $H$ always has max-degree $\beta = \Theta(1/\varepsilon^2)$, we can maintain the integral matching $M$ in $O(1/\varepsilon^4)$ time *per update to* $H$. Over the run of the algorithm, we only perform $O(\mu(G)/\varepsilon^4)$ updates to $H$ (see Lemma 7), hence the total additional update time spent maintaining the integral matching will be $O(n/\varepsilon^8)$.

**Proof of Theorem 1.** To prove Theorem 1, we need a slightly more refined analysis than the one above. We still run our incremental algorithm (Algorithm 1 and Theorem 5) to maintain a weighted EDCS $H$ together with a fractional matching supported on $H$. Similarly to above, we additionally maintain an $(1-\varepsilon)$-approximate (integral) matching $M$ of $H$.

The main idea of the fully dynamic algorithm of Gupta-Peng [34] is to lazily recompute (in $O(|H|/\varepsilon)$ time via Hoproft-Karp Theorem 4) $M$ every $\approx \varepsilon\mu$ updates to $H$ (indeed, during this few updates, the matching size cannot change its value by more than $\varepsilon\mu$). There are two observations which helps us to do better:

- **(i)** The graph $G$ (but not the sparsifier $H$) is incremental, so $\mu(G)$ can only grow.
- **(ii)** We know a good estimate of $\mu(G)$, namely the size of our fractional matching. Denote by $\tilde{\mu}$ the value of the maintained fractional matching, so that $(1-\varepsilon)\mu(G) \leq \tilde{\mu} \leq \mu(G)$.

The above two observations mean that we only need to recompute the matching $M$ whenever $\mu(G)$ actually have increased significantly (namely by a $(1+\Theta(\varepsilon))$-factor), and not just every $\varepsilon\mu$ updates.

Formally, whenever $|M| \geq (1-\varepsilon)^2\tilde{\mu}$ we know that $M$ is still a $(1-3\varepsilon)$-approximation since then $|M| \geq (1-\varepsilon)^2\tilde{\mu} \geq (1-\varepsilon)^3\mu(G) \geq (1-3\varepsilon)\mu(G)$. Conversely, whenever $|M| < (1-\varepsilon)^2\tilde{\mu}$, we recompute $M$ in time $O(|H|/\varepsilon)$ (Theorem 4) so that it is a $(1-\varepsilon)$-approximation of the maximum matching in $H$ (and thus also a $(1-2\varepsilon)$-approximation of the maximum matching in $G$).

Let us now bound the total time spent recomputing $M$. Let $M_1, M_2, \ldots, M_t$ be the different approximate matchings we compute during the run of the algorithm. We first note that at the time when we compute $M_{i+1}$:

$$|M_i| \leq (1-\varepsilon)^2\tilde{\mu} \leq (1-\varepsilon)\left((1-\varepsilon)\mu(H)\right) \leq (1-\varepsilon)|M_{i+1}| \tag{2}$$

This in turn means that $|M_i| \leq (1-\varepsilon)^{t-i}\, n$ (since $|M_t| \leq n$), and hence that $\sum_{i=1}^{t} |M_i| \leq n\sum_{i=0}^{\infty}(1-\varepsilon)^i \leq n/\varepsilon$, by a geometric sum.

Finally we note that we spend $O(|M_i|/\varepsilon^3)$ time in order to compute $M_i$. Indeed, when we compute $M_i$, we did so in $O(|H|/\varepsilon)$ time, and $|H| = O(\mu(G)/\varepsilon^2)$ by Lemma 6. This means that in total, over the run of the algorithm, we spend $O(\sum |M_i|/\varepsilon^3) = O(n/\varepsilon^4)$ time

maintaining the integral matchings $M_i$. This is in addition to the time spent maintaining the weighted EDCS $H$ and the fractional matching (see Theorem 5). This concludes the proof of Theorem 1. ◀

## 4 Vertex Deletions

In this section we will observe that our algorithm can also handle vertex deletions (simultaneously to handling edge insertions) in similar total update time. However, this comes with one caveat: we instead get additive approximation error proportional to $\varepsilon n$ (that is we maintain a matching of size $\mu^*(G) - \varepsilon n$, instead of $\mu^*(G) - \varepsilon\mu^*(G)$ as before).

▶ **Theorem 10.** *For any $\varepsilon \in (0,1)$, there is an algorithm that maintains a* fractional *matching of size at least $\mu^*(G) - \varepsilon n$ in an graph $G$ undergoing edge insertions and vertex deletions. The total update time is $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

**Proof.** The algorithm (Algorithm 1) remains the same as in Section 3.1. When a vertex is deleted, we simply remove all it's incident edges from $E$ and $H$, and call `FixVertex` on all neighboring vertices (in $H$) who now changed their degree. The only thing which changes in the analysis is the $\beta^2\mu^*(G)$-bound on the number of updates to $H$ (Lemma 7), which no longer applies. However, we can still get a weaker version of Lemma 7 with a $\beta^2 n$ total update bound instead:

▶ **Lemma 11.** *If a multiset of edges $H$ is only ever changed by (i) removing overfull edges, (ii) adding underfull edges, and (iii)* removing all edges incident to a vertex when no edges are underfull or overfull*, then there are at most $3\beta^2 n$ insertions/deletions to $H$.*

Given Lemma 11 (which we prove in Section 7), we see that the running time analysis of Algorithm 1 can remain exactly the same! In the approximation guarantee analysis, we now have more "update-heavy" vertices $|R_V| \leq 6\beta^2 n/(2\beta^2/\varepsilon) = 3\varepsilon n$, which is why we now can lose up to $O(n\varepsilon)$ edges from the matching. Otherwise, the approximation guarantee analysis remains the same, and so does the rest of the analysis of the algorithm. ◀

**Rounding in Bipartite Graphs.** Similar as in Section 3.2, we can round the fractional matching to an integral one in bipartite graphs, also while supporting edge-insertions and vertex-deletions simultaneously.

▶ **Theorem 2.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1, \epsilon \cdot n)$-approximate maximum matching of a bipartite graph undergoing edge insertions and vertex deletions in total update time of $O(n/\varepsilon^8 + m/\varepsilon^5)$.*

**Proof.** Unlike in Section 3.2, we cannot argue that $\mu(G)$ is increasing when we have vertex deletions. So instead we resort to the Gupta-Peng [34] framework discussed in Remark 9 (together with Lemma 11), which has the additional cost of $O(n/\varepsilon^8)$ total update time to maintain an approximate integral matching on our sparsifier $H$. ◀

▶ Remark 12. We note that normal vertex-sparsification techniques (such as the one shown in [38] against oblivious adversaries) do not apply here in order to assume $n = \tilde{\Theta}(\mu(G))$ so that the additive error becomes multiplicative again. This is because vertex deletions in the original graph might become edge deletions in the vertex-sparsified graph. We also note that one can achieve similar guarantees of supporting vertex deletions with additive $\varepsilon n$ slack using *any* edge-incremental algorithm (also for non-bipartite graphs) as a black-box: see Section 6 for a discussion on how this can be done. The general approach in Section 6 will give worse dependency on $1/\varepsilon$ (for dense graphs), compared to Theorems 2 and 10 above.

# 5 Tight Bounds of the of Approximation Ratio of a Weighted EDCS

## 5.1 Explicit Fractional Matching

In this section, we give explicit formulas only based on the degrees in $H$, for a $(1-\varepsilon)$-approximate fractional matching. We prove this by also providing an explicit approximate fractional vertex cover, and showing that these satisfy approximate complimentary slackness. This also significantly simplifies the previous proof [16] that $H$ is $(1-\varepsilon)$-matching sparsifier in bipartite graphs.

▶ **Theorem 13.** *Suppose $H$ is a weighted EDCS of a graph $G$, with parameter $\beta \geq 25/\varepsilon^2$. Let $f_{(u,v)} = \min\{\frac{1}{\deg_H(v)}, \frac{1}{\deg_H(u)}\}$ for each[8] $(u,v) \in H$. Then $f$ is a $(1-\epsilon)$-approximate fractional matching of $G$.*

Define $r_v := \deg_H(v) - \frac{\beta-1}{2}$ for a vertex $v \in H$. We define the fractional matching $f$ as in the statement of the theorem, together with the fractional vertex cover $x$:

$$f_{(u,v)} = \min\left(\frac{1}{\deg_H(u)}, \frac{1}{\deg_H(v)}\right) \quad \text{for edge } (u,v) \in H \tag{3}$$

$$x_v = \begin{cases} \min(1, \frac{1}{2} + \frac{r_v^2}{\beta}) & \text{if } r_v \geq 0 \\ \max(0, \frac{1}{2} - \frac{r_v^2}{\beta}) & \text{if } r_v < 0 \end{cases} \tag{4}$$

It is now relatively straightforward (albeit a bit calculation-heavy) to argue that $f$ and $x$ are indeed feasible solutions and that they satisfy approximate complimentary slackness.

▷ Claim 14. Our $f$ is a fractional matching and our $x$ is a fractional vertex cover of $G$.

Proof. Our $f$ is feasible since no vertex $v$ is overloaded by the matching: at most $\deg_H(v)$ many incident edges to $v$ contribute at most $1/\deg_H(v)$ each.

To argue that $x$ is feasible, consider some edge $(u,v) \in E$. Without loss of generality we may assume that $\deg_H(v) \geq \deg_H(u)$ and $\deg_H(v) \geq \frac{\beta-1}{2}$, i.e. $r_v \geq r_u$ and $r_v \geq 0$ (since $\deg_H(u) + \deg_H(v) \geq \beta - 1$ as $H$ is a weighted EDCS). If $r_v^2 \geq \beta/2$, $x_v = 1$ so $(u,v)$ is covered. In the case $r_v^2 < \beta/2$, we instead have $x_v = \frac{1}{2} + \frac{r_v^2}{\beta}$. It is always the case that $x_u \geq \frac{1}{2} - \frac{r_u^2}{\beta}$. Additionally we note that $r_u + r_v \geq 0$ (so $r_u^2 \leq r_v^2$) since $\deg_H(u) + \deg_H(v) \geq \beta - 1$, so we conclude that $x_u + x_v \geq 1$, and hence that $(u,v)$ is covered. ◁

▷ Claim 15. The fractional matching $f$ and fractional vertex cover $x$ satisfy $(1 - \frac{3}{\sqrt{\beta}}, 1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta})$-approximate complementary slackness[9]; in particular:

- **(i)** Whenever $f_{(u,v)} > 0$, then $x_u + x_v \leq 1 + \frac{2}{\sqrt{\beta}} + \frac{1}{\beta}$.
- **(ii)** Whenever $x_v > 0$, then $\sum_{u:(u,v)\in H} f_{(u,v)} \geq 1 - \frac{4}{\sqrt{\beta}}$.

Proof. We verify (i) and (ii).

- **(i)** Suppose $f_{(u,v)} > 0$, then $(u,v) \in H$, so $\deg_H(u) + \deg_H(v) \in \{\beta - 1, \beta\}$. This means that $0 \leq r_v + r_u \leq 1$. If both $r_v$ and $r_u$ are non-negative, we have that $x_u + x_v \leq 2(\frac{1}{2} + \frac{1^2}{\beta}) \leq 1 + \frac{2}{\beta}$. Now, without loss of generality $r_u < 0 \leq r_v$. In case

[8] We note that edges $e$ appearing multiple time in $H$ all contribute towards $f_e$: if $e$ appears $\phi_e$ times in $H$, the value of $f_e$ is naturally scaled by $\phi_e$.

[9] For completeness, we define the approximate complimentary slackness conditions in Section 7 and prove them in Lemma 20.

$r_u^2 \geq \beta/2$, we know $x_u = 0$ so $x_u + x_v \leq 1$. In the case when $r_u^2 < \beta/2$, we know $x_u = \frac{1}{2} + \frac{r_u^2}{\beta}$ and $x_v \leq \frac{1}{2} + \frac{r_v^2}{\beta}$. Since $r_u + r_v \leq 1$, we known that $r_v \leq |r_u| + 1$. Concluding:

$$x_v + x_u \leq 1 + \frac{(|r_u|+1)^2 - r_u^2}{\beta} = 1 + \frac{2|r_u|+1}{\beta} < 1 + \frac{2\sqrt{\beta}+1}{\beta} = 1 + \frac{2}{\sqrt{\beta}} + \frac{1}{\beta}.$$

**(ii)** Suppose $x_v > 0$. Hence $r_v > -\sqrt{\beta/2}$ (else $x_v = 0$), that is $\deg_H(v) > \frac{\beta-1}{2} - \sqrt{\beta/2}$. For any incident edge $(u,v) \in H$, we must have $\deg_H(v) + \deg_H(u) \leq \beta$, so $\deg_H(u) \leq \frac{\beta-1}{2} + (1 + \sqrt{\beta/2})$. Now, we see that we assign weight at least $1/(\frac{\beta-1}{2} + (1+\sqrt{\beta/2}))$ to the edge $(u,v)$ in $f$. Since this holds for all the $\deg_H(v) > \frac{\beta-1}{2} - \sqrt{\beta/2}$ incident edges we know that $v$ will in total receive, from the fractional matching $f$, at least:

$$\sum_{u:(u,v)\in H} f_{(u,v)} \geq \frac{\frac{\beta-1}{2} - \sqrt{\beta/2}}{\frac{\beta-1}{2} + 1 + \sqrt{\beta/2}} = 1 - \frac{\sqrt{8\beta}+2}{\beta + \sqrt{2\beta}+1} \geq 1 - \frac{3}{\sqrt{\beta}}.$$ ◀

**Proof of Theorem 13.** By the above claims and approximate complimentary slackness (see Lemma 20 in Section 7) we know that $(1 - \frac{3}{\sqrt{\beta}})|x| \leq (1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta})|f|$. Since $(1 - \frac{3}{\sqrt{\beta}})/(1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta}) > 1 - \frac{5}{\sqrt{\beta}}$, we get that $|f| \geq (1 - \frac{5}{\sqrt{\beta}})|x| \geq (1-\varepsilon)|x| \geq (1-\varepsilon)\mu^*(G)$ whenever $\beta \geq 25/\varepsilon^2$. ◀

## 5.2 Lower Bound

In this section we show that Theorem 13 is tight up to a constant, i.e. that one must set $\beta = \Theta(1/\varepsilon^2)$ in order to guarantee that a weighted EDCS preserves a $(1-\varepsilon)$-fraction of the matching. This might be a bit surprising, considering that for the *unweighted* EDCS, it is known that $\beta = \Theta(1/\varepsilon)$ suffices (to preserve a $(\frac{2}{3} - \varepsilon)$-approximation to the matching [7]).

▶ **Theorem 16.** *For any $\beta \geq 2$, there exists a (bipartite) graph $G$ together with a weighted EDCS $H$ for which $\mu(H) = (1 - \Theta(1/\sqrt{\beta}))\mu(G)$.*

We show our construction in Figure 1, and also describe it here formally in words. For simplicity, we will assume that $\beta = 2\gamma^2$ for some integer $\gamma$ (but it is not difficult to adapt the proof for when $\beta$ is not twice a square). In our construction, each edge appears at most once in $H$, and all edges $e \in H$ have $\deg_H(e) = \beta$; all edges $e \in E \setminus H$ have $\deg_H(e) = \beta - 1$.

Define the gadget $G_i = (S_i, L_i, E_i)$ to be a complete bipartite graph in which $|S_i| = \gamma^2 + i$ and $|L_i| = \gamma^2 - i$ ($S$ is for vertices with *small* degree, and $L$ for vertices with *large* degree). The subgraph $H$ will consist of many of these gadgets $G_i$, so we start by noting a few properties about them. Firstly, vertices in $L_i$ have degree $\frac{\beta}{2} + i$ while those in $S_i$ have degree $\frac{\beta}{2} - i$. This means that any edge in $(u,v) \in E_i$ has degree exactly $\deg_{G_i}(u) + \deg_{G_i}(v) = \beta$. We also note that the maximum matching inside $G_i$ is of size $|L_i| = \gamma^2 - i$.

**Subgraph $H$.** We begin by describing how the weighted EDCS $H$ looks like, and later we will define what additional edges are also in the full graph $G$. The subgraph $H$ will exactly consist of:

- One copy of $G_0$.
- Two copies each of $G_1, G_2, \ldots, G_{\gamma-1}$. Call the copies $G_i^{(1)}$ and $G_i^{(2)}$.
- $2\gamma$ copies of $G_\gamma$. Call the copies $G_\gamma^{(1,j)}$ and $G_\gamma^{(2,j)}$ for $j = 1, 2, \ldots, \gamma$.

▷ Claim 17. $\mu(H) = 4\gamma^3 - 4\gamma^2 + \gamma$.

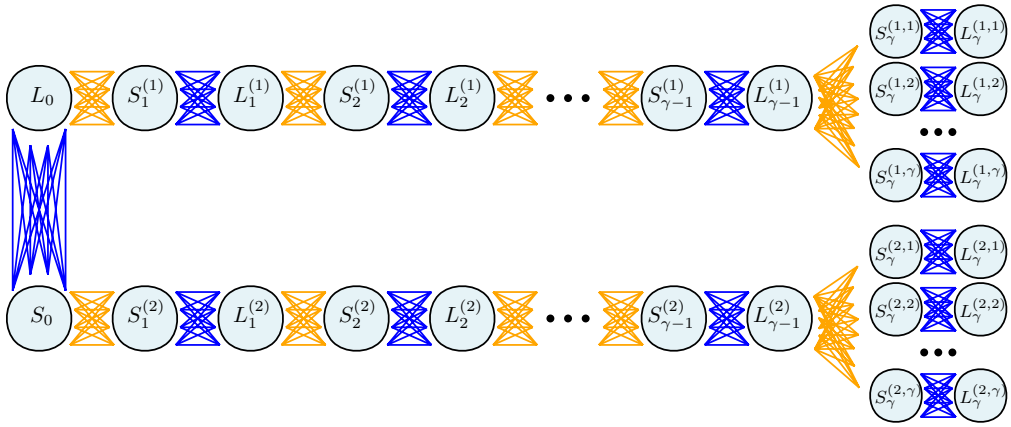



**Figure 1** The lower bound construction. The blue edges are part of $H$, while the yellow are not. We have $\gamma = \sqrt{\beta/2}$, and each set $S_i$, $L_i$ indicates an independent set of vertices of size $|S_i| = \gamma^2 + i$ and $|L_i| = \gamma^2 - i$ (so in $H$ they have degrees $\frac{\beta}{2} - i$ and $\frac{\beta}{2} + i$ respectively). The maximum matching in $H$ matches all vertices in $L_i$ to the corresponding $S_i$. The maximum matching in $G$ however, matches $L_i$ to $S_{i+1}$ (and $S_0$ to $S_1^{(2)}$), in addition to $L_\gamma$ which can also be matched to $S_\gamma$.

Proof. Since the maximum matching size in $G_i$ is $|L_i| = \gamma^2 - i$ we get:

$$\begin{aligned}\mu(H) &= |L_0| + 2\gamma|L_\gamma| + 2(|L_1| + |L_2| + \cdots + |L_{\gamma-1}|) \\ &= \gamma^2 + 2\gamma(\gamma^2 - \gamma) + 2\sum_{i=1}^{\gamma-1}(\gamma^2 - i) \\ &= 4\gamma^3 - 4\gamma^2 + \gamma\end{aligned}$$

◁

**Full graph $G$.** Now we describe the additional edges which are part of $G$ but not already in $H$ (see also Figure 1):

- For $k \in \{1, 2\}$, we connect $G_1^{(k)}, G_2^{(k)}, \ldots, G_{\gamma-1}^{(k)}$ in a chain as follows: every pair $(u, v)$ with $u \in L_i^{(k)}$ and $v \in S_{i+1}^{(k)}$ is an edge (so that the induced subgraph on these two sets of vertices forms a complete bipartite graph).
- At the end of these two chains, we connect all the gadgets $G_\gamma^{(k,j)}$ as follows: every pair $(u, v)$ with $u \in L_{\gamma-1}^{(k)}$ and $v \in S_\gamma^{(k,j)}$ for some $j$, is an edge.
- Finally we connect these two chains using $G_0 = (S_0, L_0, E_0)$ as follows: every pair $(u, v)$ with $u \in L_0$ and $v \in S_1^{(1)}$ is an edge; and every pair $(u, v)$ with $u \in S_0$ and $v \in S_1^{(2)}$ is an edge.

We note that $G$ is bipartite and all above edges have degree exactly $\deg_H(u) + \deg_H(v) = \beta - 1$, so indeed $H$ is a weighted EDCS of $G$.

▷ Claim 18. $\mu(G) \geq 4\gamma^3 - 3\gamma^2 + \gamma$.

Proof. We argue that a matching of this size exists in $G$. In fact the only edges of $H$ we will use as part of this matching are those in the gadgets $G_\gamma^{(k,j)}$.

- We pick a matching between $L_i^{(k)}$ and $S_{i+1}^{(k)}$ of size $|L_i^{(k)}| = \gamma - i$ for all $k \in \{1, 2\}$ and $i = 1, 2, \ldots, \gamma - 2$.
- In $G_\gamma^{(k,j)}$ we pick a matching of size $|L_\gamma^{(k,j)}| = \gamma^2 - \gamma$. Note that exactly $2\gamma$ vertices in $S_\gamma^{(k,j)}$ are left unmatched.

- Denote by $U^{(k)}$ the set of unmatched vertices in $S_\gamma^{(k,1)}, S_\gamma^{(k,2)}, \ldots, S_\gamma^{(k,\gamma)}$, for $k \in \{1,2\}$. Note that $|U^{(k)}| = 2\gamma^2$ and that $(L_{\gamma-1}^{(k)}, U^{(k)})$ forms a complete bipartite graph, so we pick a matching of size $|L_{\gamma-1}^{(k)}| = \gamma^2 - \gamma + 1$ from there.
- Finally we pick matchings between $L_0$ and $S_1^{(1)}$ (respectively $S_0$ and $S_1^{(2)}$) of size $|L_0| = \gamma$ (respectively $|S_0| = \gamma$).

In total we see that the above matching is exactly $|S_0| = \gamma$ larger than in Claim 17, which concludes the proof of the claim. ◁

**Approximation ratio.** To conclude the proof of Theorem 16, we see that $\mu(G) - \mu(H) = \gamma^2 \geq \frac{1}{4\gamma}\mu(G)$ whenever $\gamma \geq 1$. Hence $H$ preserves at most a $(1 - \frac{1}{4\gamma}) = (1 - \frac{1}{2\sqrt{2\beta}})$ fraction of the maximum matching of $G$.

## 6 Black-Box Vertex Deletions

Here we briefly explain how one can convert any incremental $(1-\varepsilon)$-approximate maximum matching algorithm to also support vertex deletions, if allowing additive $\varepsilon n$ approximation instead, in a black-box fashion. The reduction is simple and also works in general (non-bipartite) graphs. Hence, as an immediate application, we can get a $(1, \varepsilon n)$-approximate matching algorithm for general graphs undergoing both edge-insertions and vertex-deletions, with amortized $1/\varepsilon^{O(1/\varepsilon)}$ update time, if using the incremental algorithm of [31].

▶ **Lemma 19.** *Suppose $\mathcal{A}$ is an algorithm which maintains a $(1-\varepsilon/2)$-approximate maximum matching for a graph undergoing edge insertions, running in total time $T$. Then there exists an algorithm which maintains a $(1, \varepsilon n)$-approximate matching, in total time $O(T/\varepsilon)$, on a graph undergoing both edge insertions and vertex deletions.*

**Proof.** We run $\mathcal{A}$, and whenever we get a vertex deletion we ignore it and keep the vertex in the graph. In the outputted matching from the algorithm we remove any edges incident to deleted vertices. When $\varepsilon n/2$ vertices have been deleted, we actually delete them from the graph and rerun the algorithm from scratch (starting on the empty graph). This will only happen $\frac{2}{\varepsilon}$ times, which is the running time blow-up. At each point, the algorithm maintains a matching of size at least $\mu - (\varepsilon n/2 + \varepsilon n/2) = \mu - \varepsilon n$, since only one edge can be removed per deleted vertex still remaining in the graph. ◀

## 7 Omitted Proofs

### EDCS properties

▶ **Lemma 6.** *In a $\beta$-WEDCS $H$, the maximum degree is at most $\beta$ and $|H| \leq \beta\mu^*(G)$.*

**Proof.** If a vertex $u$ has $\deg_H(u) > \beta$, then any incident edge $(u,v) \in H$ is overfull: $\deg_H(u) + \deg_H(v) > \beta$, leading to a contradiction. Hence the maximum degree is at most $\beta$. Now we construct a fractional matching by assigning a weight of $1/\beta$ to every edge in $H$ (so an edge appearing with multiplicity $\phi$ in $H$ gets weight $\phi/\beta$). Clearly this is a feasible fractional matching of $G$, since no vertex is overloaded. On the other hand, the size of this fractional matching is $|H|/\beta$, implying that $|H| \leq \beta\mu^*(G)$. ◀

▶ **Lemma 7.** *If a multiset of edges $H$ is only ever changed by removing overfull edges and adding underfull edges, then there are at most $\beta^2\mu^*(G)$ such insertions/deletions to $H$.*

**Proof.** We use a potential function argument. Define

$$\Phi(H) := |H|(2\beta-1) - \sum_{(u,v)\in H} (\deg_H(v)+\deg_H(u)) = \sum_{(u,v)\in H} (2\beta-1-\deg_H(u)-\deg_H(v))$$

We note that if an edge $(u,v)$ appears multiple times in $H$, it appears multiple times in the above sum as well. We first note that $\Phi(\emptyset)=0$ and $\Phi(H) \leq \beta|H| \leq \beta^2\mu^*(G)$, by Lemma 6 and since $(2\beta-1-\deg_H(u)-\deg_H(v)) \leq \beta$ when $H$ is a valid EDCS. Now we verify that updates to $H$ increase the potential by at least 1:

- Insertion of an underfull edge $(u,v) \in E$.
  That is $\deg_H(u)+\deg_H(v) \leq \beta-2$ before adding the edge. The term $|H|(2\beta-1)$ will increase by $2\beta-1$. $\sum_{(u,v)\in H}(\deg_H(v)+\deg_H(u))$ will increase by at most $2\beta-2$, since one term of value $\deg_H(v)+\deg_H(v) \leq \beta-2+2$ (the $+2$ comes from $\deg_H(v)$ and $\deg_H(u)$ increasing by one when we add $(u,v)$ to $H$) is added, and at most $\beta-2$ other terms decrease in value by one.
- Deletion of an overfull edge $(u,v) \in H$.
  That is $\deg_H(u)+\deg_H(v) \geq \beta+1$ before removing the edge. The term $|H|(2\beta-1)$ will decrease by $2\beta-1$. $\sum_{(u,v)\in H}(\deg_H(v)+\deg_H(u))$ will decrease by at least $2\beta$, since one term of value $\deg_H(v)+\deg_H(v) \geq \beta+1-2$ (the $-2$ comes from $\deg_H(v)$ and $\deg_H(u)$ decreasing when we remove $(u,v)$) is erased, and at least $\beta+1$ other terms increase in value by one. ◀

▶ **Lemma 11.** *If a multiset of edges $H$ is only ever changed by (i) removing overfull edges, (ii) adding underfull edges, and (iii)* removing all edges incident to a vertex when no edges are underfull or overfull*, then there are at most $3\beta^2 n$ insertions/deletions to $H$.*

**Proof.** We continue the potential function argument from the proof of Lemma 7 above. When we delete, from $H$, all edges incident to some vertex $u$, we know that we deleted at most $\beta$ many edges from $H$ (as the degree of this vertex was at most $\beta$). For each such incident edge $(u,v)$, we bound how much its deletion could have decreased the potential function. The $|H|(2\beta-1)$ term decreased by exactly $2\beta-1$, and the $-\sum_{(u,v)\in H}(\deg_H(u)+\deg_H(v))$ term can only increase. So the total decrease in potential, over all up to $\beta$ incident edges which were deleted, is at most $2\beta^2-\beta$.

Since we can only delete up to $n$ vertices in total, and the potential is always bounded by $\beta^2\mu^*(G) \leq \beta^2 n$, it follows that the total increase in the potential function, over the run of the algorithm, is at most $3\beta^2 n - n\beta$ (and thus this many updates to $H$ from insertions/deletions of underfull/overfull edges). In total we deleted at most $n\beta$ edges in $H$ incident to deleted vertices, so the total number of updates to $H$ is thus bounded by $3\beta^2 n - \beta n + \beta n = 3\beta^2 n$. ◀

### Approximate Complimentary Slackness

▶ **Lemma 20.** *Suppose we have the primal linear program $\max\{c^T x : Ax \leq b, x \geq 0\}$, and its dual $\min\{b^T y : A^T y \geq c, y \geq 0\}$. We say that feasible primal solution $x$ and dual solution $y$ satisfy $(\alpha,\gamma)$-approximate complementary slackness (for $\alpha \leq 1 \leq \gamma$) if: (i) if $x_i = 0$ then $(A^T)_i y \leq \gamma c_i$, and (ii) if $y_j = 0$ then $(A)_j x \geq \alpha b_j$. When this is the case, then $\alpha b^T y \leq \gamma c^T x$ (i.e. $x$ and $y$ are $\frac{\gamma}{\alpha}$-approximate optimal).*

**Proof.** We see that $\gamma c^T x - \alpha b^T y = x^T(\gamma c - A^T y) + y^T(Ax - \alpha b)$. Now either $x_i = 0$ or $(\gamma c - A^T y)_i \geq 0$; and either $y_j = 0$ or $(Ax-\alpha b)_j \geq 0$. Hence $\gamma c^T x - \alpha b^T y \geq 0$. ◀

## References


1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *STOC*, pages 114–125. ACM, 2019. doi:10.1145/3313276.3316376.

2 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *ICALP*, volume 107 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.7.

3 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *SODA*, pages 1616–1635. SIAM, 2019. doi:10.1137/1.9781611975482.98.

4 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *ICALP*, volume 198 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.19.

5 Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *SOSA*, volume 69 of *OASIcs*, pages 11:1–11:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.11.

6 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in O(log n) update time (corrected version). *SIAM J. Comput.*, 47(3):617–650, 2018. Announced at FOCS'11. doi:10.1137/16M1106158.

7 Soheil Behnezhad. Improved analysis of EDCS via gallai-edmonds decomposition. *CoRR*, abs/2110.05746, 2021.

8 Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *SODA*, pages 129–162. SIAM, 2023. doi:10.1137/1.9781611977554.CH6.

9 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, pages 382–405. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00032.

10 Soheil Behnezhad and Sanjeev Khanna. New trade-offs for fully dynamic matching via hierarchical EDCS. In *SODA*, pages 3529–3566. SIAM, 2022. doi:10.1137/1.9781611977073.140.

11 Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully dynamic matching: Beating 2-approximation in $\Delta^\epsilon$ update time. In *SODA*, pages 2492–2508. SIAM, 2020. doi:10.1137/1.9781611975994.152.

12 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In *STOC*, pages 267–280. ACM, 2023. doi:10.1145/3564246.3585231.

13 Aaron Bernstein. Improved bounds for matching in random-order streams. In *ICALP*, volume 168 of *LIPIcs*, pages 12:1–12:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.12.

14 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. *ACM Trans. Algorithms*, 17(4):29:1–29:51, 2021. Announced at SODA'19. doi:10.1145/3469833.

15 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *FOCS*, pages 1123–1134. IEEE, 2020. doi:10.1109/FOCS46700.2020.00108.

16 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. doi:10.1007/978-3-662-47672-7_14.

17 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *SODA*, pages 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.CH50.

18 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *IPCO*, volume 10328 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 2017. doi:10.1007/978-3-319-59250-3_8.

19 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018. Announced at SODA'15. doi:10.1137/140998925.

20 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, pages 398–411. ACM, 2016. doi:10.1145/2897518.2897568.

21 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, pages 470–489. SIAM, 2017. doi:10.1137/1.9781611974782.30.

22 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *ICALP*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.27.

23 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. In *SODA*, pages 1–47. SIAM, 2023. doi:10.1137/1.9781611977554.CH1.

24 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for $(1.5+\epsilon)$-approximate matching. In *STOC*, pages 254–266. ACM, 2023. doi:10.1145/3564246.3585252.

25 Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *SODA*, pages 100–128. SIAM, 2023. doi:10.1137/1.9781611977554.CH5.

26 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $O(1/\epsilon^2)$ amortized update time. In *SODA*, pages 1872–1885. SIAM, 2019. doi:10.1137/1.9781611975482.113.

27 Joakim Blikstad and Peter Kiss. Incremental (1-(epsilon))-approximate dynamic matching in O(poly(1/(epsilon))) update time. *CoRR*, abs/2302.08432, 2023. doi:10.48550/arXiv.2302.08432.

28 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *ICALP*, volume 107 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.33.

29 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *FOCS*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

30 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.

31 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *SODA*, pages 1886–1898. SIAM, 2019. doi:10.1137/1.9781611975482.114.

32 Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. Maintaining an EDCS in general graphs: Simpler, density-sensitive and with worst-case time bounds. In *SOSA*, pages 12–23. SIAM, 2022. doi:10.1137/1.9781611977066.2.

33 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, volume 29 of *LIPIcs*, pages 227–239. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICS.FSTTCS.2014.227.

34 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *FOCS*, pages 548–557. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.65.

35 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

**36** John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

**37** Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Regularized box-simplex games and dynamic decremental bipartite matching. In *ICALP*, volume 229 of *LIPIcs*, pages 77:1–77:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.77.

**38** Peter Kiss. Deterministic dynamic matching in worst-case update time. In *ITCS*, volume 215 of *LIPIcs*, pages 94:1–94:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.94.

**39** Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955.

**40** Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *FOCS*, pages 248–255. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.40.

**41** Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. Announced at STOC'13. doi:10.1145/2700206.

**42** Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464. ACM, 2010. doi:10.1145/1806689.1806753.

**43** David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: A density-sensitive approach. In *SODA*, pages 712–729. SIAM, 2016. doi:10.1137/1.9781611974331.CH51.

**44** Mohammad Roghani, Amin Saberi, and David Wajc. Beating the folklore algorithm for dynamic matching. In *ITCS*, volume 215 of *LIPIcs*, pages 111:1–111:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.111.

**45** Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126. SIAM, 2007.

**46** Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, pages 325–334. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.43.

**47** David Wajc. Rounding dynamic matchings against an adaptive adversary. In *STOC*, pages 194–207. ACM, 2020. doi:10.1145/3357713.3384258.

**48** Da Wei Zheng and Monika Henzinger. Multiplicative auction algorithm for approximate maximum weight bipartite matching. In *IPCO*, volume 13904 of *Lecture Notes in Computer Science*, pages 453–465. Springer, 2023. doi:10.1007/978-3-031-32726-1_32.

# Maximum Independent Set When Excluding an Induced Minor: $K_1 + tK_2$ and $tC_3 \uplus C_4$

**Édouard Bonnet**
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

**Julien Duron**
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

**Colin Geniet**
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

**Stéphan Thomassé**
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

**Alexandra Wesolek**
Simon Fraser University, Burnaby, Canada

## Abstract

Dallard, Milanič, and Štorgel [arXiv '22] ask if, for every class excluding a fixed planar graph $H$ as an induced minor, Maximum Independent Set can be solved in polynomial time, and show that this is indeed the case when $H$ is any planar complete bipartite graph, or the 5-vertex clique minus one edge, or minus two disjoint edges. A positive answer would constitute a far-reaching generalization of the state-of-the-art, when we currently do not know if a polynomial-time algorithm exists when $H$ is the 7-vertex path. Relaxing tractability to the existence of a quasipolynomial-time algorithm, we know substantially more. Indeed, quasipolynomial-time algorithms were recently obtained for the $t$-vertex cycle, $C_t$ [Gartland et al., STOC '21], and the disjoint union of $t$ triangles, $tC_3$ [Bonamy et al., SODA '23].

We give, for every integer $t$, a polynomial-time algorithm running in $n^{O(t^5)}$ when $H$ is the friendship graph $K_1+tK_2$ ($t$ disjoint edges plus a vertex fully adjacent to them), and a quasipolynomial-time algorithm running in $n^{O(t^2 \log n)+f(t)}$, with $f$ a single-exponential function, when $H$ is $tC_3 \uplus C_4$ (the disjoint union of $t$ triangles and a 4-vertex cycle). The former generalizes the algorithm readily obtained from Alekseev's structural result on graphs excluding $tK_2$ as an induced subgraph [Alekseev, DAM '07], while the latter extends Bonamy et al.'s result.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** Maximum Independent Set, forbidden induced minors, quasipolynomial-time algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.23

**Funding** This work was supported by the ANR projects TWIN-WIDTH (ANR-21-CE48-0014) and Digraphs (ANR-19-CE48-0013).
*Alexandra Wesolek*: Supported by the Vanier Canada Scholarship Program.

**Acknowledgements** We would like to thank Dibyayan Chakraborty for useful discussions, and anonymous reviewers for their helpful comments, and in particular a nice simplification in the proof of Theorem 2.

## 1 Introduction

The Max Independent Set (MIS for short) problem asks, in its optimization form, for a largest *independent set* of its input graph $G$, i.e., a subset of pairwise non-adjacent vertices in $G$. In its decision form, the input is a graph $G$ and an integer $k$, and the question is whether $G$ admits an independent set of size at least $k$.

Besides the ubiquitous usefulness that such a fundamental problem has within combinatorial optimization, and notably in the areas of packing, scheduling, and coloring, MIS (or equivalently Maximum Clique, the same problem in the complement graph) has as wide a range of applications as possible, as evidenced for instance in map labeling [48], coding theory [17], spatial scheduling [26], genetic analysis [1], information retrieval [10], macromolecular docking [28], and sociometry [27] (also see Butenko's thesis [16]). It is thus unfortunate that this problem is not only hard to solve but also very reluctant to being approximated. Indeed, the decision version of MIS is NP-complete [29] and W[1]-complete [25], while its optimization version cannot be approximated within ratio $n^{1-\varepsilon}$ on $n$-vertex graphs, for any $\varepsilon > 0$, unless P=NP [35, 50].

In spite of this, theorists and practitioners have put a lot of effort in designing efficient algorithms for MIS. In parallel to generically solving MIS via integer programming, highly performant exact and heuristic MIS solvers have emerged in the recent years, based on diverse methods such as kernelization and evolutionary approaches [40], deep reinforcement learning [4], graph neural networks [44], and dataless training (where backpropagation is applied to a loss function based instead on the input) [9]. On the theory side, exact exponential algorithms have been developed for decades culminating in a running time below $1.2^n n^{O(1)}$ [49].

Another approach is to try and exploit the structure that the input graphs may have. Indeed, in all the aforementioned applications, inputs are not uniformly sampled over all $n$-vertex graphs: They instead bare some structural properties, and in some cases, might avoid some specific patterns. Graph theory[1] offers two main notions of *patterns* or containment: the natural and straightforward *subgraphs* (obtained by removing vertices and edges), and the deeper *minors* (further allowing to contract edges). Both notions come with an induced variant, when edge removals are disallowed, bringing the number of containment types to four. It is then sensible to determine the patterns $H$ whose absence makes MIS (more) tractable. It turns out that this question is completely settled for subgraphs and minors.

For the subgraph containment, the argument is the following. By the grid minor theorem [45], the class of graphs excluding $H$ as a subgraph has bounded treewidth if (and only if) all the connected components of $H$ are paths and subdivided *claws* (i.e., stars with three leaves); thus MIS can be solved in polynomial-time in this class, for instance by Courcelle's theorem [21]. If instead $H$ has a connected component which is not a path nor a subdivided claw, MIS remains NP-complete since such a class either contains all subcubic graphs or the $2|V(H)|$-subdivision of every graph, two families of graphs on which MIS is known to be NP-complete [5, 8, 43].

For minors, the dichotomy relies on the planarity of $H$. Indeed, if $H$ is planar, then the class of graphs excluding $H$ as a minor has bounded treewidth (again, mainly by the grid minor theorem), and MIS can be solved efficiently. If $H$ is non-planar, then the $H$-minor-free graphs include all planar graphs for which MIS is known to be NP-complete [30].

The question is more intriguing for the induced containments, and the *induced subgraph* case has received a lot of attention. While it is known for a long time that if $H$ is *not* the disjoint union of paths and subdivided claws, MIS remains NP-complete on graphs without $H$ as an induced subgraph [5, 43], it has been conjectured that MIS is otherwise polynomial-time solvable. This has been proven when $H$ is the 6-vertex path [34], a claw with exactly one edge subdivided [6, 41], or any disjoint union of claws [15]. The latter result extends a polynomial-time algorithm (essentially) due to Alekseev when $H$ is any disjoint

[1] We refer the reader to Section 2 for the relevant background in graph theory.

union of edges [7]. The author indeed proves that the total number of maximal independent sets is polynomially bounded. One can then enumerate all the maximal independent sets in polynomial time (following Alekseev's proof, or using the generic output-sensitive algorithm of Tsukiyama et al. [47]), and thus find a maximum independent set.

While we currently do not know of a polynomial-time algorithm when $H$ is the 7-vertex path, Gartland and Lokshtanov [31] have obtained a quasipolynomial-time algorithm when $H$ is $P_t$, the $t$-vertex path, for any positive integer $t$; also see [42]. Supporting the existence of at least a quasipolynomial-time algorithm when $H$ is $S_{i,j,k}$, the claw whose three edges are subdivided $i-1$, $j-1$, and $k-1$ times, respectively, a quasipolynomial-time approximation scheme (QPTAS) [20] and a polynomial-time algorithm among bounded-degree graphs [2] have been proposed. The parameterized complexity of MIS when excluding a fixed induced subgraph has been studied [12, 13, 22], but the mere statement of which $H$ make the problem fixed-parameter tractable (and which ones keep it W[1]-complete) is unclear [13].

We eventually arrive at the *induced minor* containment, the topic of the current paper. As for minors, the class of all graphs excluding a non-planar graph $H$ as an induced minor contains all planar graphs; hence MIS remains NP-complete in such a class. However we do not know of a *planar* graph $H$, for which MIS remains NP-complete on $H$-induced-minor-free graphs. This has led Dallard, Milanič, and Štorgel [23] to ask if such classes exist:

▶ **Question 1.** *Is it true that for every planar graph $H$, Max Independent Set can be solved in polynomial time in the class of graphs excluding $H$ as an induced minor?*

A first observation is that avoiding $H$ as an induced minor implies avoiding it as an induced subgraph. Thus Question 1 is settled for $P_6$, $S_{1,1,2}$, and $tS_{1,1,1}$ (where $tG$ denotes the disjoint of $t$ copies of $G$). The same authors [23] further obtain a polynomial-time algorithm when $H$ is $K_5^-$ (the 5-vertex clique minus an edge), $K_{2,t}$ (the bipartite complete graph with 2 vertices fully adjacent to $t$ vertices), and $W_4 = K_1 + C_4$ (a 4-vertex cycle $C_4$ with a fifth vertex fully adjacent to the cycle). All three cases were shown by bounding the so-called *tree-independence number* (i.e., treewidth where *bag size* is replaced by *independence number of the subgraph induced by the bag*) [23], in which case a polynomial-time algorithm can be derived for MIS using the corresponding tree-decompositions [24]. They also show that this is as far as this sole technique can go: $H$-induced-minor-free graphs have bounded *tree-independence number* if and only if $H$ is edgeless or an induced minor of $K_5^-$, $K_{2,t}$, or $W_4$ [23]. The framework of *potential maximal cliques* [14] and the *container method* has led to a polynomial-time algorithm when $H = C_5$ [3, 19].

Question 1 is a beautiful question and, if true, a very difficult one. Indeed, $H = P_7$, the 7-vertex path, is a very simple planar graph for which we currently do not know such a polynomial-time algorithm. A natural relaxation of Question 1 is to only request a quasipolynomial-time algorithm:

▶ **Question 2.** *Is it true that for every planar graph $H$, Max Independent Set can be solved in quasipolynomial time in the class of graphs excluding $H$ as an induced minor?*

We know somewhat more about Question 2. There is a quasipolynomial-time algorithm for MIS in $C_t$-induced-minor-free graphs [32], building upon the $H = P_t$ case. Recently, Bonamy et al. [11] present a quasipolynomial-time algorithm when $H$ is $tC_3$, i.e., the disjoint union of $t$ triangles. See Table 1 for a summary of the introduction.

Expecting an affirmative solution to Question 1 or Question 2 may seem optimistic. However, as far as precise running time is concerned, we do know that for every planar $H$, MIS is probably not as difficult in $H$-induced-minor-free graphs as it is in general graphs.

**Table 1** The complexity of Max Independent Set when $H$ is excluded as one of the four main types of patterns. "$\neg tS_{t,t,t}$" means that $H$ is not a subgraph of $tS_{t,t,t}$ for any $t$. Our results are framed.

| $H$ exluded as | subgraph | minor | induced subgraph | **induced minor** |
|---|---|---|---|---|
| in P | $tS_{t,t,t}$ | planar | $P_6$, $S_{1,1,2}$, $tS_{1,1,1}$ | $K_5^-$, $K_{2,t}$, $W_4$, $C_5$, $\boxed{K_1+tK_2}$ |
| known in QP | – | – | $P_t$ $(t \geqslant 7)$ | $C_t$ $(t \geqslant 6)$, $\boxed{tC_3 \uplus C_4}$ |
| NP-c | $\neg tS_{t,t,t}$ | non-planar | $\neg tS_{t,t,t}$ | non-planar |
| open P / NP-c | – | – | $P_7$, ... | $P_7$, $C_6$, ... |
| open QP / NP-c | – | – | $S_{1,1,3}$, $S_{1,2,2}$ ... | $C_4 \uplus C_4$, ... |

Indeed, Korhonen [39] describes a $2^{O(n/\log^c n)}$-time algorithm, for some constant $c > 0$, to solve MIS on $n$-vertex graphs excluding a fixed planar graph $H$ as an induced minor. Assuming the Exponential-Time Hypothesis [36], such a running time is impossible in general graphs [37].

**Our results.** We make some progress regarding Questions 1 and 2. Our first contribution is, for every positive integer $t$, a polynomial-time algorithm when $H$ is the *friendship graph* $K_1 + tK_2$ (also called *Dutch windmill graph* or *fan*), i.e., $t$ independent edges universally linked to a $2t + 1$-st vertex:

▶ **Theorem 1.** *For every positive integer $t$, Max Independent Set can be solved in polynomial-time $n^{O(t^5)}$ in $n$-vertex $K_1 + tK_2$-induced-minor-free graphs.*

This extends Alekseev's result [7] for graphs excluding $tK_2$ as an induced subgraph, or equivalently, as an induced minor. We indeed use this result to first derive a polynomial-time algorithm in subgraphs of $K_1 + tK_2$-induced-minor-free graphs $G$ induced by vertices from a bounded number of breadth-first search (BFS) layers of $G$.

We then consider the connected components of our input graph $G$ when deprived of a subset $X$ of vertices inducing $tK_2$ and, subject to that property, maximizing the order of the largest connected component in $G - X$. We show that, due to this careful selection of $X$, every component $C$ of $G - X$ admits an efficiently constructible path-decomposition $\mathcal{P}$ with *bounded adhesion* (i.e., any two distinct bags have a bounded intersection), each bag of which is contained in a bounded number of consecutive BFS layers of $C$. Hence MIS can be solved efficiently within a bag, by our opening step (see previous paragraph). This part is quite technical, but mostly to justify the existence of $\mathcal{P}$. The algorithm itself remains simple.

Theorem 1 is then obtained by exhaustively finding $X$ and guessing its intersection $X'$ with a maximum independent set of $G$, and performing dynamic programming on the connected components of $G - X$, deprived of $N(X')$. The dynamic-programming table is filled via the efficient algorithm when handling an induced subgraph contained in few BFS layers.

Our second contribution is a quasipolynomial-time algorithm when $H$ is, $tC_3 \uplus C_4$, the disjoint union of $t$ triangles and a 4-vertex cycle:

▶ **Theorem 2.** *For every positive integer $t$, Max Independent Set can be solved in quasipolynomial-time $n^{O(t^2 \log n)+f(t)}$ (where $f$ is single-exponential) in $tC_3 \uplus C_4$-induced-minor-free graphs.*

We first perform a quasipolynomial branching rule to get rid of holes of size at most 6 (i.e., induced cycles of length 4, 5, or 6). We then assume that the input graph $G$ is not $(t+2)C_3$-induced-minor-free, for otherwise we conclude with Bonamy et al.'s algorithm [11]. Thus $G$, being $tC_3 \uplus C_4$-induced-minor-free, has to admit $(t+2)C_3$ as an induced subgraph, i.e., a collection $T_1, \dots, T_{t+2}$ of $t+2$ pairwise vertex-disjoint and non-adjacent triangles. We define $S_{i,j}$, minimally separating $T_i$ and $T_j$ in the graph $G$ deprived of the neighborhoods of the other triangles $T_k$ (with $k \neq i, j$).

We show that each $S_{i,j}$ induces a clique. So does every intersection $N_{i,j}$ of the neighborhood of two distinct triangles $T_i, T_j$ of the collection (this is where getting rid of the holes of length at most 6 comes into play). We can therefore exhaustively guess the intersection of a maximum independent set with the union of the sets $S_{i,j}$ and $N_{i,j}$ (for every $i < j \in [t+2]$). We finally observe that $G' = G - \bigcup_{i \neq j \in [t+2]} (S_{i,j} \cup N_{i,j})$ is chordal, since the presence of a hole $H$ in $G'$ would imply the existence in $G$ of $t$ independent triangles in the non-neighborhood of $H$, a contradiction to the $tC_3 \uplus C_4$-induced-minor-freeness of $G$. We thus conclude by using a classic algorithm for MIS in chordal graphs [33, 46].

In Section 2 we introduce the relevant graph-theoretic background. In Section 3 we prove Theorem 1, and in Section 4 we prove Theorem 2.

## 2 Preliminaries

If $i \leqslant j$ are two integers, we denote by $[i, j]$ the set of integers $\{i, i+1, \dots, j-1, j\}$, and by $[i]$, the set $[1, i]$. We denote by $V(G)$ and $E(G)$ the set of vertices and edges of a graph $G$, respectively. We denote by $G_1 \simeq G_2$ the fact that the two graphs $G_1$ and $G_2$ are *isomorphic*, i.e., equal up to renaming their vertex set. For $S \subseteq V(G)$, the *subgraph of $G$ induced by $S$*, denoted $G[S]$, is obtained by removing from $G$ all the vertices that are not in $S$ (together with their incident edges). Then $G - S$ is a short-hand for $G[V(G) \setminus S]$. A graph $H$ is an *induced subgraph* of $G$ if there is an $S \subseteq V(G)$ such that $G[S] \simeq H$.

For $G$ a graph and a set $X \subseteq V(G)$, $E_G(X)$ (or simply $E(X)$) is a short-hand for $E(G[X])$. For $G$ a graph and $X, Y \subseteq V(G)$ two disjoint sets, $E_G(X, Y)$ denotes the set of edges of $E(G)$ with one endpoint in $X$ and the other endpoint in $Y$. We denote by $N_G(v)$ and $N_G[v]$, the open, respectively closed, neighborhood of $v$ in $G$. For $S \subseteq V(G)$, we set $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$ and $N_G[S] := N_G(S) \cup S$. We may omit the subscript if $G$ is clear from the context. A *connected component* is a maximal connected induced subgraph.

Two cycles $C, C'$ are said to be *independent* if they are vertex-disjoint and there is no edge between $C$ and $C'$. A collection of cycles is *independent* if they are pairwise independent. Two vertex subsets $X, Y \subseteq V(G)$ *touch* if $X \cap Y \neq \emptyset$ or there is an edge $uv \in E(G)$ with $u \in X$ and $v \in Y$. Then two (or more) cycles are independent if and only if they do *not* touch. We say that $X, Y \subseteq V(G)$ *touch in* $Z$ if $X \cap Y \cap Z \neq \emptyset$ or there is an edge $uv \in E(G)$ with $u \in X \cap Z$ and $v \in Y \cap Z$, or equivalently, if $X \cap Z$ and $Y \cap Z$ touch.

A graph $H$ is an *induced minor* of a graph $G$ if $H$ can be obtained from $G$ by a sequence of vertex deletions and edge contractions. A *minor* is the same but also allows edge deletions. Equivalently an induced minor $H$ –with vertex set, say, $\{v_1, \dots, v_{V(H)}\}$– of $G$ can be defined as a vertex-partition $B_1, \dots, B_{|V(H)|}$ of an induced subgraph of $G$, such that every $G[B_i]$ is connected and $v_i v_j \in E(H)$ if and only if $E_G(B_i, B_j) \neq \emptyset$ (i.e., when the disjoint sets $B_i$ and $B_j$ touch). Observe indeed that contracting each $B_i$ into a single vertex (which is possible since each $B_i$ induces a connected subgraph) results in $H$. A graph $G$ (resp. a graph class) is said to be *$H$-induced-minor-free* if $H$ is not an induced minor of $G$ (resp. no graph of the class admits $H$ as an induced minor).

We denote by $C_\ell$ the $\ell$-vertex cycle, and by $K_\ell$, the $\ell$-vertex complete graph. A *hole* is an induced cycle of length at least four. A graph is *chordal* if it has no hole. For two disjoint sets $X, Y \subseteq V(G)$ in a graph $G$, an $(X,Y)$-*separator* is a (possibly empty) set $S \subseteq V(G) \setminus (X \cup Y)$ such that there is no path between $X$ and $Y$ in $G - S$. An $(X,Y)$-separator is *minimal* if no proper subset of it is itself an $(X,Y)$-separator.

The *disjoint union* $G_1 \uplus G_2$ of two graphs $G_1, G_2$ has vertex set $V(G_1) \uplus V(G_2)$ and edge set $E(G_1) \uplus E(G_2)$, where $V(G_1) \uplus V(G_2)$ presupposes that the vertex sets of $G_1$ and $G_2$ are disjoint. If $t \geqslant 2$ is an integer and $G$ a graph, $tG$ is the graph $G \uplus (t-1)G$, and $1G$ is simply $G$. The *join* $G_1 + G_2$ of two graphs $G_1, G_2$ has vertex set $V(G_1) \uplus V(G_2)$ and edge set $E(G_1) \uplus E(G_2) \uplus \{uv \ : \ u \in V(G_1), v \in V(G_2)\}$. In other words, the join of $G_1$ and $G_2$ is obtained from their disjoint union by adding all possible edges between $G_1$ and $G_2$.

A *breadth-first search* (BFS) *layering* in $G$ from a vertex $v \in V(G)$ (or from a connected set $S \subseteq V(G)$) is a partition of the remaining vertices into $L_1, L_2, \ldots$ such that every vertex of $L_i$ is at distance exactly $i$ from $v$ (or from $S$). Such an $L_i$ is called a *BFS layer* of $G$ (from $v$, or from $S$). Note that there cannot be an edge in $G$ between $L_i$ and $L_j$ if $|i-j| > 1$.

A *path-decomposition* of a graph $G$ is a list of vertex subsets $\mathcal{P} = (B_1, \ldots, B_h)$ such that

- $\bigcup_{1 \leqslant i \leqslant h} B_i = V(G)$,
- for every $e \in E(G)$, there is some $B_i$ that contains both endpoints of $e$, and
- whenever $v \in B_i \cap B_j$ with $i < j$, $v$ is also in all $B_k$ with $i < k < j$.

The sets $B_i$ (for $i \in [h]$) are called the *bags* of $\mathcal{P}$, and the sets $B_i \cap B_{i+1}$ (for $i \in [h-1]$) the *adhesions* of $\mathcal{P}$. Path-decomposition $\mathcal{P}$ has maximum adhesion $p$ if all of its adhesions have size at most $p$. Note that the adhesion $B_i \cap B_{i+1}$, if disjoint from $B_1 \cup B_h$, is a vertex cutset disconnecting $B_1$ from $B_h$.

## 3 Polynomial algorithm in $K_1 + tK_2$-induced-minor-free graphs

We first show how to solve Max Independent Set in $K_1 + tK_2$-induced-minor-free graphs of bounded diameter. More generally, we show the following.

▶ **Lemma 3.** *Let $t, h$ be fixed non-negative integers. Let $G$ be a $K_1 + tK_2$-induced-minor-free $n$-vertex graph, and $L_0 \subseteq V(G)$ such that $G[L_0]$ is connected. Let $L_i$, for any $i \in [h]$, be the subset of vertices of $G$ at distance exactly $i$ of $L_0$. Then, given as input $G$, $L_0$, and $S \subseteq \bigcup_{1 \leqslant i \leqslant h} L_i$, a maximum independent set of $H := G[S]$ can be computed in polynomial time $n^{(2t-1)h+O(1)}$.*

**Proof.** For every $j \in [h]$, $G[L_j]$ has no $tK_2$ induced subgraph (or equivalently, induced minor). Indeed, $G[\bigcup_{0 \leqslant i \leqslant j-1} L_i]$ is a connected graph, hence $\bigcup_{0 \leqslant i \leqslant j-1} L_i$ can be contracted to a single vertex, and every vertex in $L_j$ has at least one neighbor in $L_{j-1}$. Therefore a $tK_2$ induced subgraph in $G[L_j]$ would contradict the $K_1 + tK_2$-induced-minor-freeness of $G$.

Fix an arbitrary $S \subseteq \bigcup_{1 \leqslant i \leqslant h} L_i$, and consider the induced subgraph $H := G[S]$. In particular $H[L_j \cap S]$ has also no $tK_2$ induced subgraph, for every $j \in [h]$. Hence, by a classical result of Alekseev [7], $H[L_j \cap S]$ has at most $n^{2t-1}$ maximal independent sets, which can be listed in time $n^{2t+O(1)}$ [47].

We thus exhaustively list every $h$-tuple $(I_1, \ldots, I_h)$ where, for every $j \in [h]$, $I_j$ is a maximal independent set of $H[L_j \cap S]$, in time $n^{(2t-1)h+O(1)}$. Note that if there is an edge in $H$ between $L_i$ and $L_j$, then $|i-j| \leqslant 1$. As each $I_j$ (for $j \in [h]$) is an independent set, $H' = H[\bigcup_{j \in [h]} I_j]$ is a bipartite graph as witnessed by the bipartition $(I_1 \cup I_3 \cup \ldots, I_2 \cup I_4 \cup \ldots)$. A maximum independent set $I$ can thus be computed in polynomial time in $H'$. Indeed, by the Kőnig-Egerváry theorem [38], finding a maximum independent set in a bipartite graph boils down

to finding a maximum matching, which can be done in polynomial time (and now, even in almost linear time [18]) by solving a maximum flow problem. We output the largest independent set $I$ found among every run.

The correctness of the algorithm is based on the observation that a maximum independent set $I^\star$ of $H$ intersects every $L_i$ (for $i \in [h]$) in an independent set $J_i$, which by definition is contained in a maximal independent set $I_i$ of $H[L_i \cap S]$. In the run when every maximal independent set $I_i$ is a superset of $J_i$, we obtain an independent set with cardinality equal to that of $I^\star$. ◀

We say that $G$ is *reduced* if it does not contain degree-1 vertices. If $e = uv$ is an edge of $G$, let $G \setminus e$ (resp. $S \setminus e$, $S \cup e$, for some $S \subseteq V(G)$) be the induced subgraph $G[V(G) - \{u, v\}] = G - \{u, v\}$ (resp. the sets $S \setminus \{u, v\}$, $S \cup \{u, v\}$). More generally, for a collection $e_1 = u_1v_1, \ldots, e_k = u_kv_k$ of edges of $G$, we denote by $G \setminus \{e_1, \ldots, e_k\}$ the induced subgraph $G[V(G) - \{u_1, v_1, \ldots, u_k, v_k\}] = G - \{u_1, v_1, \ldots, u_k, v_k\}$.

▶ **Lemma 4.** *Let $G$ be a reduced connected $K_1 + tK_2$-induced-minor-free graph containing $tK_2$ as an induced subgraph. Let $X \subseteq V(G)$ maximize the order of a largest component of $G' := G - X$, among those sets $X$ such that $G[X] \simeq tK_2$. Then for any $e \in E(G' - N_G(X))$ contained in a connected component $C$ of $G'$,*

1. *$C \setminus e$ is disconnected, and*
2. *each connected component of $C \setminus e$ contains a vertex in $N_G(X)$.*

**Proof.** As $G$ is reduced, every vertex of $X$ has degree at least two (in $G$). Thus $G'$ cannot be connected, for otherwise, contracting in $G$ the set $V(G')$ to a single vertex would form a $K_1 + tK_2$ induced minor. We thus know that $G'$ has at least two connected components.

Let $C'$ be a largest connected component of $G'$. Since $G$ is connected, there exists a shortest path $P$ in $G$ from $V(C')$ to $V(G') \setminus V(C')$. Say, that $P$ ends in the connected component $C \neq C'$ of $G'$. Path $P$ has to have some internal vertices in $X$, but since $G[X] \simeq tK_2$, it follows that there is an edge $e^*$ in $E(X)$ (but not necessarily in $P$) intersecting both $N_G(V(C))$ and $N_G(V(C'))$.

For every edge $e \in E(G' - N_G(X))$ in component $C$ (which is possibly equal to $C'$), $C \setminus e$ is disconnected. Indeed, for the sake of contradiction, suppose that $C \setminus e$ is connected, and consider $X' := (X \setminus e^*) \cup e$. By assumption, $G[X'] \simeq tK_2$. Furthermore, the connected component of $G - X'$ containing $e^*$ is strictly larger than $C'$, as it contains $(V(C') \setminus e) \cup e^*$ and intersects $V(C) \setminus e$, which are two disjoint sets. This contradicts the maximality of $X$, and establishes the first item.

We now prove the second item, also by contradiction. Suppose that there is a connected component $D$ of $C \setminus e$ that does not contain a vertex in $N_G(X)$. We will reach a contradiction by showing that $D$ contains an edge $e'$ not intersecting $N_G(X)$, and such that $D \setminus e'$ is connected (and conclude in light of the previous paragraph).

Let $L_i \subseteq V(D)$ be the *$i$-th neighborhood of $e$* in $D$, i.e., the vertices at distance $i$ of one endpoint of $e$, and at least $i$ of the other endpoint. We consider the *last layer* $L_k$, i.e., such that $L_k \neq \emptyset$ and $L_{k+1} = \emptyset$. If $L_k$ contains an edge $e'$, then removing the endpoints of this edge does not disconnect $D$ (and hence $C$) since each vertex in $L_k$ has a neighbor in $L_{k-1}$ (with the convention that $L_0$ consists of the endpoints of $e$) and $G[L_0 \cup L_1 \cup \cdots \cup L_{k-1}]$ is connected.

If $L_k$ does not contain an edge, then each vertex in $L_k$ has two neighbors in $L_{k-1}$. This is because $G$ is reduced, and by assumption that no vertex of $D$ has a neighbor in $X$. Hence, removing the endpoints of any edge $e'$ incident to a vertex in $L_k$ does not disconnect $D$ (nor $C$), since each vertex in $L_k$ has at least one neighbor in $L_{k-1}$ which is not an endpoint of $e'$. In either case, $e'$ is an edge of $C - N_G(X)$ that does not disconnect $C$, which we showed is not possible. ◀

We now prove the main technical result of the section.

▶ **Proposition 5.** *Let $G$ be a reduced connected $n$-vertex $K_1 + tK_2$-induced-minor-free graph containing $tK_2$ as an induced subgraph. Let $X \subseteq V(G)$ maximize the order of a largest component of $G' := G - X$, among those sets $X$ such that $G[X] \simeq tK_2$. Then for every connected component $C$ of $G'$, a path-decomposition $\mathcal{P}$ of $C$ such that*

- *every bag of $\mathcal{P}$ is contained in $O(t^4)$ consecutive BFS layers of $C$, and*
- *every adhesion of $\mathcal{P}$ is of size at most $2t^2$,*

*can be computed in time $n^{O(1)}$.*

**Proof.** Let $v$ be a vertex in $N_G(X) \cap V(C)$ and, for any positive integer $s$, let $L_s$ be the set of vertices at distance $s$ from $v$ in $C$. Let $q$ be the largest distance between $v$ and a vertex of $C$. We set $f(t) := (t^2+1)(6t^2+2) = O(t^4)$. We will show that, for every $s \in [q - f(t)]$, there is a vertex cutset of size at most $2t^2$ separating $L_s$ from $L_{s+f(t)-1}$, and use that fact for $s = 1, f(t)+1, 2f(t)+1, \ldots$ to build the path-decomposition $\mathcal{P}$.

We show that any sufficiently long induced path (such as a shortest path from $L_s$ to $L_{s+f(t)-1}$) has some edges with both endpoints in $V(C) \setminus N_G(X)$.

▷ Claim 6. Any induced path $P$ in $C$ contains less than $3t^2$ vertices in $N_G(X)$.

Proof. If there are $3t^2$ vertices on $P$ with a neighbor in $X$, then there are at least $3t$ vertices $w_1, \ldots, w_{3t}$ on $P$ that are neighbors of a fixed edge $e \in E(X)$. For each $i \in [t]$, contract every edge of $P$ between $w_{3i-2}$ and $w_{3i-1}$ but one, say $e_i$. Contract $e$, and call $z$ the resulting vertex. The vertex $z$ and the $t$ edges $e_i$ contradict the fact that $G$ is $K_1 + tK_2$-induced-minor-free. ◁

For an edge $e \in E(C)$ we denote by $\mathrm{dist}(e, v)$ the length of a shortest path in $C$ from an endpoint of $e$ to $v$. We build a collection of paths $Q_i$ of $C$, and edges $e_i \in E(Q_i)$, for $i = 1, 2, \ldots$, while they are well-defined, in the following way.

Let $s \in [q - f(t)]$ and $Q_1$ be a shortest path from $L_s$ to $L_{s+f(t)-1}$ in $C$. Let $e_1 \in E(Q_1)$ minimize $e \mapsto \mathrm{dist}(e, v)$, among those edges of $Q_1$ with both endpoints in $V(C) \setminus N_G(X)$. By Claim 6 there are less than $6t^2$ edges on $Q_1$ with an endpoint in $N_G(X)$, hence $\mathrm{dist}(e_1, v) \leqslant s + 6t^2$. We denote by $Q_1'$ the maximal subpath of $Q_1$ starting in $L_s$ and not containing an endpoint of $e_1$ (that is, stopping just before reaching an endpoint of $e_1$). Note that $Q_1$ is possibly empty. For the next iteration, we work in $C \setminus e_1$ (recall that this stands for $C$ deprived of the two endpoints of $e_1$).

We now describe in general the $i$-th iteration for $i \geqslant 2$. Let $Q_i$ be a shortest path from $L_s$ to $L_{s+f(t)-1}$ in $C \setminus \{e_1, e_2, \ldots, e_{i-1}\}$. Let $e_i$ be the first edge of $Q_i$ (when starting from $L_s$) such that $\mathrm{dist}(e_i, v) \geqslant \mathrm{dist}(e_{i-1}, v) + 2$ and $e_i$ has no endpoint in $N_G(X)$. Note that by Claim 6, $\mathrm{dist}(e_i, v) \leqslant \mathrm{dist}(e_{i-1}, v) + 2 + 6t^2$. Let $Q_i'$ be the maximal subpath of $Q_i$ starting in $L_s$ and not containing an endpoint of $e_i$. See Figure 1.

Let $e_1, \ldots, e_k$ be the obtained collection of edges. In principle, the while loop stops when one of the following two conditions holds:

- **(i)** $L_s$ is disconnected from $L_{s+f(t)-1}$ in $C \setminus \{e_1, \ldots, e_k\}$, or
- **(ii)** there is no edge $e_{k+1} \in E(Q_{k+1})$ such that $\mathrm{dist}(e_{k+1}, v) \geqslant \mathrm{dist}(e_k, v) + 2$ and $e_{k+1}$ has no endpoint in $N_G(X)$.

▷ Claim 7. If case $(ii)$ holds, then $k > t^2$.

Proof. Remark first that for any $i$ we have $\mathrm{dist}(e_i, v) \leqslant s + i(6t^2+2) - 2$. Next, we claim that as long as $\mathrm{dist}(e_i, v) + 2 + 6t^2 \leqslant s + f(t) - 1$, and case $(i)$ does not occur, the construction of $e_{i+1}$ can not fail, since $Q_{i+1}$ exists, and there are at least $6t^2$ edges $e$ in $Q_{i+1}$ satisfying $\mathrm{dist}(e, v) \geqslant \mathrm{dist}(e_i, v)$. Thus if case $(ii)$ occurs, it must be that $(k+1)(6t^2+2) - 2 > f(t) - 1$, hence $k > t^2$ since $f(t) = (t^2+1)(6t^2+2)$. ◁

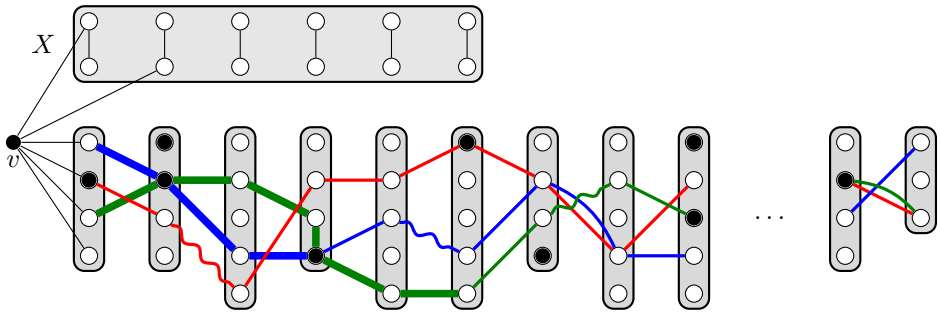



**Figure 1** Illustration of $X$ (for $t = 6$) and the first (i.e., with $s = 1$) $f(t)$ layers (from left to right) of the connected component $C$ of $G - X$ rooted at $v$. Vertices of $C$ with a neighbor in $X$ are filled. We represented the first three iterations: $Q_1, e_1$ (in red), $Q_2, e_2$ (in blue), $Q_3, e_3$ (in dark green). Not to clutter the figure, we do not represent all the edges, and we code the different labels: edges $e_i$ are squiggly, and subpaths $Q'_i$ are thicker.

▷ Claim 8. It holds that $k \leqslant t^2$.

Proof. Assume for the sake of contradiction that $k \geqslant t^2 + 1$. By Lemma 4 (using both items), for each $i \in [k]$ there exists a vertex $v_i \in V(C) \cap N_G(X)$ disconnected from $v$ in $C \setminus e_i$. Since $k \geqslant t^2 + 1$, there are $t + 1$ vertices $v_{a_1}, \ldots, v_{a_{t+1}}$, with $a_1 < a_2 < \ldots < a_t$, all adjacent to an endpoint of the same fixed edge $e \in E(X)$. Let $C'$ be the component containing $v$ in $C \setminus \{e_{a_1}, \ldots, e_{a_t}\}$, and let us contract (in $G$) the set $V(C')$ to a single vertex, say $w$. We will now show that vertex $w$ is adjacent to an endpoint of $e$.

By construction, the edge $e_{a_{t+1}}$ is reachable from $v$ in $C \setminus \{e_{a_1}, e_{a_2}, \ldots, e_{a_t}\}$, hence $e_{a_{t+1}} \in E(C')$. Let $P'$ be a path in $C'$ from $v$ to an endpoint of $e_{a_{t+1}}$. Every path in $C$ from $v$ to $v_{a_{t+1}}$ goes through at least one endpoint of $e_{a_{t+1}}$, by definition of $v_{a_{t+1}}$. Let us consider a shortest path in $C$ from $v$ to $v_{a_{t+1}}$, i.e., intersecting each layer at most once. Consider a suffix $P''$ of this path starting at an endpoint of $e_{a_{t+1}}$ and ending at $v_{a_{t+1}}$. By the previous remark, $P''$ cannot contain an edge among $\{e_{a_1}, e_{a_2}, \ldots, e_{a_t}\}$, since these edges are in layers with strictly smaller indices than the endpoints of $e_{a_{t+1}}$. Thus $P' \cup P''$ (possibly combined with $e_{a_{t+1}}$) connects $v$ to $v_{a_{t+1}}$ in $C \setminus \{e_{a_1}, e_{a_2}, \ldots, e_{a_t}\}$. Therefore, the contracted vertex $w$ *contains* $v_{a_{t+1}}$; the latter being adjacent to an endpoint $e$.

Observe also that, for every $i \in [t]$, each subpath $Q'_{a_i}$ is contained in $C'$. Vertex $w$ is adjacent to an endpoint $u_{a_i}$ of $e_{a_i}$, for every $i \in [t]$. Let $C_{a_i}$ be the vertex set of the connected component of $C \setminus e_{a_i}$ containing $v_{a_i}$. Let $R_{a_i}$ be a path from $u_{a_i}$ to $v_{a_i}$ in the subgraph of $G$ induced by $C_{a_i}$ plus the endpoints of $e_{a_i}$. For each $i \in [t]$, if $R_{a_i}$ has at least two edges, contract the edge $wu_{a_i}$, and all the edges of $R_{a_i}$ but the last two; the last one being $f_{a_i} = y_{a_i} v_{a_i}$. Finally contract $e$, and the edge between $e$ and $w$. We call the resulting vertex $z$.

We claim that $z$ and the edges $f_{a_i}$ make a $K_1 + tK_2$ induced minor in $G$. One can see that $z$ is adjacent to $v_{a_i}$ via $e \in E(X)$, and to $y_{a_i}$ via $w$ and the path $R_{a_i}$. We shall justify that the edges $f_{a_i}$ form an induced matching in $G$. Indeed, suppose some $f_{a_i}$ and $f_{a_j}$ touch with $i \neq j \in [t]$. Then in $C \setminus e_{a_i}$, there is a path from $v$ to $v_{a_i}$ via $Q'_j$, a contradiction. ◁

By Claims 7 and 8, case (*ii*) is impossible, and the $2k \leqslant 2t^2$ endpoints of $e_1, \ldots, e_k$ form a vertex cutset disconnecting $L_s$ from $L_{s+f(t)-1}$. We can now build the path-decomposition $\mathcal{P}$. Recall that the BFS search from $v$ gives rise to $q$ layers, $L_1, \ldots, L_q$ (outside $\{v\}$).

For $j \in [\lfloor q/f(t) \rfloor]$, let $S_j$ be the vertex cutset of size at most $2t^2$ (and obtained as detailed above) disconnecting $L_{(j-1)f(t)+1}$ from $L_{jf(t)}$. We denote by $L_{s \to s'}$ the set $\bigcup_{s \leqslant h \leqslant s'} L_h$.

- Let $B_1 \subseteq V(C)$ consist of $v$, $S_1$, plus all the vertices of connected components of $C[L_{1\to f(t)}] - S_1$ that do not intersect $L_{f(t)}$.
- For $j$ going from 2 to $\lfloor q/f(t) \rfloor - 1$, let $B_j \subseteq V(C)$ consist of $S_j \cup S_{j+1}$ plus the vertices not already present in one of $B_1, \ldots, B_{j-1}$ of all the connected components of

$$C[L_{(j-1)f(t)+1\to(j+1)f(t)}] - (S_j \cup S_{j+1})$$

  that do not intersect $L_{(j+1)f(t)}$.
- Let finally $B_{\lfloor q/f(t)\rfloor} \subseteq V(C)$ consist of $S_{\lfloor q/f(t)\rfloor}$ plus the vertices of all the connected components of $C[L_{(\lfloor q/f(t)\rfloor-1)f(t)+1\to q}] - S_{\lfloor q/f(t)\rfloor}$ that intersect $L_q$.

Let $\mathcal{P}$ be the path-decomposition $(B_1, B_2, \ldots, B_{\lfloor q/f(t)\rfloor})$. $\mathcal{P}$ is indeed a path-decomposition of $C$, since our process entirely covers $V(C)$, and by virtue of $S_j$ separating $L_{(j-1)f(t)+1}$ from $L_{jf(t)}$. By construction,

- every bag intersects at most $2f(t) = O(t^4)$ layers of BFS from vertex $v$, and
- for every $j \in [\lfloor q/f(t)\rfloor - 1]$, $B_j \cap B_{j+1} = S_j$, so every adhesion has size at most $2t^2$.

Finally note that once $X$ is found, one can find the path-decomposition $\mathcal{P}$ of $C$ in time $n^{O(1)}$, since this only involves computing (at most $n$) shortest paths. ◀

We can now wrap up, using Proposition 5 and Lemma 3.

▶ **Theorem 1.** *For every positive integer $t$, Max Independent Set can be solved in polynomial-time $n^{O(t^5)}$ in $n$-vertex $K_1 + tK_2$-induced-minor-free graphs.*

**Proof.** Let $G$ be our $K_1 + tK_2$-induced-minor-free $n$-vertex input graph. As including vertices of degree 1 in the independent set is a safe reduction rule, we can assume that $G$ is reduced. By dealing with the possibly several connected components of $G$ separately, we can further assume that $G$ is connected. If $G$ has no $tK_2$ as an induced subgraph, we conclude by invoking Alekseev's result [7, 47]. Thus we also assume that $G$ has such an induced subgraph. In time $n^{O(t)}$ we find $X \subseteq V(G)$ that maximizes the order of a largest component of $G' := G - X$, among those sets $X$ such that $G[X] \simeq tK_2$.

We exhaustively guess the intersection $X'$ of a fixed maximum independent set of $G$ with the set $X$, with an extra multiplicative factor of $2^{2t}$. We are now left with solving MIS separately in $C' := C - N_G(X')$ for each connected component $C$ of $G'$. By Proposition 5, we obtain in time $n^{O(1)}$ a path-decomposition $\mathcal{P} = (B_1, \ldots, B_p)$ of $C'$, such that

- every $B_i$ (for $i \in [p]$) is contained in $O(t^4)$ consecutive BFS layers *of* $C$, and
- every adhesion $A_i := B_i \cap B_{i+1}$ (for $i \in [p-1]$) is of size at most $2t^2$.

Indeed, removing $N_G(X')$ from $C$ (and its path-decomposition) preserves those properties.

Let us define $A_0, A_p$ to be empty. We proceed to the following dynamic programming. For $i \in [0, p]$, and for any $S \subseteq A_i$, $T[i, S]$ is meant to eventually contain an independent set $I$ of $C'[\bigcup_{1\leqslant j\leqslant i} B_j]$ of maximum cardinality among those such that $I \cap B_i = S$. We set $T[0, \emptyset] = \emptyset$, and observe that it is the only entry of the form $T[0, \cdot]$.

We fill this table by increasing value of $i = 1, 2, \ldots, p$. Assume that all entries of the form $T[i', \cdot]$ are properly filled for $i' < i$. For every $S \subseteq A_i$, $T[i, S]$ is filled in the following way. For every $S' \subseteq A_{i-1}$, if $S \cup S'$ is an independent set, we compute, by Lemma 3 (with $L_1, \ldots, L_h$ being the $O(t^4)$ consecutive BFS layers *of* $C$ containing $B_i$, and $L_0$ being the connected set, *in* $C$, formed by the union of all the previous layers), a maximum independent set $I_i$ in $C'[B_i] - N[S \cup S']$ in time $n^{O(ht)} = n^{O(t^5)}$. We finally set $T[i, S] = T[i-1, S'] \cup I_i \cup S$ for a run that maximizes the cardinality of $T[i-1, S'] \cup I_i$.

It takes time $p \cdot 2^{O(t^2)} \cdot n^{O(t^5)} = n^{O(t^5)}$ to completely fill $T$. Eventually $T[p, \emptyset]$ contains a maximum independent set of $C'$. We return the union of $X'$ and of the maximum independent sets of $C'$ found for each connected component $C$ of $G'$. The overall running time is $n^{O(t^5)}$. ◀

## 4 Quasipolynomial algorithm in $tC_3 \uplus C_4$-induced-minor-free graphs

We first show that the existence of a short hole allows for a quasipolynomial-time branching rule in $tC_3 \uplus C_4$-induced-minor-free graphs. By branching rule, we mean here a Turing reduction to (quasipolynomially many) subinstances with no short holes.

▶ **Lemma 9.** *Let $G$ be an $n$-vertex $tC_3 \uplus C_4$-induced-minor-free graph, and let $\ell \geqslant 4$ be a fixed integer. While there is a hole $H$ of length at most $\ell$ and a $tC_3$ as an induced subgraph in $G$, Max Independent Set admits a quasipolynomial branching rule, running in $n^{O_\ell(t \log n)}$.*

**Proof.** Let $C(G) := \{X \in \binom{V(G)}{3t} : G[X] \simeq tC_3\}$ be the collection of vertex subsets inducing $t$ disjoint triangles, and assume $\mu(G) := |C(G)| > 0$, and $H$ is a hole of $G$ of length at most $\ell$. As $G$ is $tC_3 \uplus C_4$-induced-minor-free, $N[V(H)]$ intersects every $X \in C(G)$. In particular, there is a vertex $v \in V(H)$ such that $N[v]$ intersects at least a $1/\ell$ fraction of the $X \in C(G)$.

We branch on two options: either we take $v$ in an (initially empty) solution, and remove its closed neighborhood from $G$, or we remove $v$ from $G$ (without adding it to the solution). With the former choice, the measure $\mu$ drops by at least a $1/\ell$ fraction (and the number of vertices of $G$ decreases by at least 1), and with the latter choice, the number of vertices drops by 1. This branching is exhaustive. We simply need to argue about its running time.

Note that each option can be done at most $n$ times, while the first option cannot be done more than $\log_\ell(n^{3t}) = O_\ell(t \log n)$ times. Hence the branching tree has at most $\binom{n}{\log_\ell(n^{3t})} = n^{O_\ell(t \log n)}$ leaves. ◀

The previous lemma permits us to get rid of short holes, which turns out useful in some corner case.

▶ **Theorem 2.** *For every positive integer $t$, Max Independent Set can be solved in quasipolynomial-time $n^{O(t^2 \log n)+f(t)}$ (where $f$ is single-exponential) in $tC_3 \uplus C_4$-induced-minor-free graphs.*

**Proof.** We apply the quasipolynomial branching rule of Lemma 9 with $\ell = 6$, until the input $n$-vertex graph $G$ no longer has holes of length at most 6, or $tC_3$ induced subgraph.

In time $n^{O(t)}$, we exhaustively look for a collection of pairwise vertex-disjoint and non-adjacent triangles $T_1, T_2, \ldots, T_{t+2}$ in $G$. If such a collection does not exist, $G$ is $(t+2)C_3$-induced-minor-free. Indeed, the absence of $(t+2)C_3$ as an induced subgraph implies that at least one of the $(t+2)$ independent cycles realizing a $(t+2)C_3$ induced minor is of length at least four. This is ruled out by the assumption that $G$ is $tC_3 \uplus C_4$-induced-minor-free. (Note here that only $(t+1)$ independent cycles would suffice.)

We can thus assume that a collection $T_1, T_2, \ldots, T_{t+2}$ exists, for otherwise, we can conclude with the quasipolynomial-time algorithm, running in $n^{O(t^2 \log n)+f(t)}$ (where $f$ is single-exponential), of Bonamy et al. [11] for Max Independent Set in graphs with a bounded number of independent cycles (here, $(t+2)C_3$-induced-minor-free). In turn, as $G$ contains $tC_3$ (even $(t+2)C_3$) as in induced subgraph, we can, in light of the first paragraph, further assume that all the cycles of $G$ have length either 3 or at least 7. We refer the reader to Figure 2 for a visual summary of the next two paragraphs.

For every pair $T_i, T_j$ (with $i < j \in [t+2]$), consider the subgraph $G_{i,j} := G - \bigcup_{k \in [t+2] \setminus \{i,j\}} N(T_k)$. We claim that $G_{i,j}$ is chordal. Indeed, since $G_{i,j}$ is disjoint from the neighborhood of $\bigcup_{k \in [t+2] \setminus \{i,j\}} T_k$, a hole $H$ in $G$ would form a $tC_3 \uplus C_4$-induced-minor together with $\{T_k : k \in [t+2] \setminus \{i,j\}\}$. Let now $S_{i,j}$ be a minimal $(T_i, T_j)$-separator in $G_{i,j}$. A classical argument then shows that $S_{i,j}$ is a clique: suppose for the sake of contradiction that $u, v \in S_{i,j}$ are distinct and non-adjacent. Let $X_i, X_j$ be the two components of $G_{i,j} - S_{i,j}$

containing $T_i, T_j$. By minimality of $S_{i,j}$, each of $u, v$ is adjacent to both $X_i$ and $X_j$. Thus we can find two induced $uv$-paths, whose internal vertices are in $X_i$ and $X_j$ respectively. The union of these paths is a hole $H$ in $G_{i,j}$.

Let $N_{i,j}$ be the set $N(T_i) \cap N(T_j)$ for each pair $i < j \in [t+2]$. Observe that the sets $N_{i,j}$ need not be disjoint, and that when $T_i$ and $T_j$ are at distance at least 3 apart, $N_{i,j}$ is empty. We notice that $N_{i,j}$ is a clique, for otherwise we can exhibit an induced cycle of length 4, 5, or 6 in $G$ (hence a hole of length at most 6).

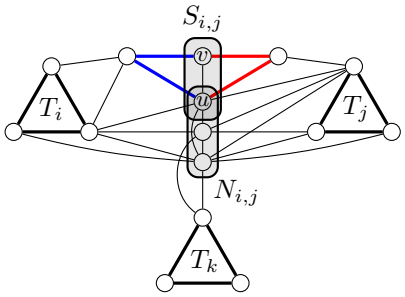



**Figure 2** Illustration of the sets $S_{i,j}$ and $N_{i,j}$, and the two $uv$-paths through the two components, in red and blue. If $uv$ were a non-edge, these paths would form a hole in the non-neighborhood of the other triangles $T_k$, contradicting $tC_3 \uplus C_4$-minor-freeness. The absence of hole of length at most 6 implies that $N_{i,j}$ is also a clique.

We claim that $G' := G - (\bigcup_{i<j\in[t+2]} S_{i,j} \cup N_{i,j})$ is chordal. Indeed assume there is a hole $H'$ in $G'$. The $tC_3 \uplus C_4$-induced-minor-freeness implies that $H'$ intersects at least two sets $N[T_i]$ and $N[T_j]$. Thus there exists a subpath $P$ of $H'$ whose endpoints are in two distinct $N(T_i)$ and $N(T_j)$. By choosing $P$ minimal, we can furthermore assume that no internal vertex of $P$ lies in some $N[T_k]$ with $k \notin \{i, j\}$. Since $G'$ does not include any vertex of $\bigcup_{i'<j'\in[t+2]} N_{i',j'}$, the endpoints of $P$ are not in some $N[T_k]$ with $k \notin \{i, j\}$ either. Therefore, the path $P$ contradicts that $S_{i,j}$ separates $T_i$ and $T_j$ in $G - \bigcup_{k\in[t+2]\setminus\{i,j\}} N(T_k)$.

We can now describe the rest of the algorithm after the collection $T_1, T_2, \ldots, T_{t+2}$ is found. We greedily compute the minimal separators $S_{i,j}$. We exhaustively try every subset $S \subseteq \bigcup_{i<j\in[t+2]} S_{i,j} \cup N_{i,j}$ that is an independent set. Such a set $S$ contains at most one vertex in each $S_{i,j}$ and each $N_{i,j}$, as we have established that each $S_{i,j}$ and each $N_{i,j}$ form a clique. Hence there are $n^{O(t^2)}$ such sets $S$. For each $S$, we compute a maximum independent set $I$ in the *chordal* graph $G - (N[S] \cup \bigcup_{i<j\in[t+2]} S_{i,j} \cup N_{i,j})$ in linear time (see [33, 46]). We finally output the set $S \cup I$ maximizing $|S \cup I|$. Note that the overall running time is $n^{O(t^2 \log n)+f(t)}$. ◀

## 5 Conclusion

We provided a polynomial-time algorithm for Maximum Independent Set on graphs excluding the friendship graph as an induced minor and a quasipolynomial-time algorithm on graphs excluding a disjoint union of $t$ triangles and a 4-cycle as an induced minor. As mentioned in the introduction, it is of interest to study on which other classes of graphs exluding some fixed planar graph $H$ as an induced minor MIS can be solved in (quasi)polynomial-time. If $H$ is a subgraph of the friendship graph or of the wheel, some of our methods developed for $K_1 + tK_2$ might extend.

For disjoint unions of cycles, the first open case is when $H$ is $C_4 \uplus C_4$. Our treatment for $H = tC_3 \uplus C_4$ does not (easily) extend to this case. It is thus likely that new methods have to be found. Obtaining a quasipolynomial-time algorithm when $H = tC_t$ for any integer $t$ is a first challenging milestone in the study of MIS on $H$-induced-minor-free graphs.

## References


1 Kuruvilla J. Abraham and Clara Diaz. Identifying large sets of unrelated individuals and unrelated markers. *Source code for biology and medicine*, 9(1):1–8, 2014.

2 Tara Abrishami, Maria Chudnovsky, Cemil Dibek, and Paweł Rzazewski. Polynomial-time algorithm for maximum independent set in bounded-degree graphs with no long induced claws. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 1448–1470. SIAM, 2022. doi:10.1137/1.9781611977073.61.

3 Tara Abrishami, Maria Chudnovsky, Marcin Pilipczuk, Paweł Rzążewski, and Paul D. Seymour. Induced subgraphs of bounded treewidth and the container method. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021*, pages 1948–1964. SIAM, 2021. doi:10.1137/1.9781611976465.116.

4 Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 134–144. PMLR, 2020. URL: http://proceedings.mlr.press/v119/ahn20a.html.

5 Vladimir E. Alekseev. The effect of local constraints on the complexity of determination of the graph independence number. *Combinatorial-algebraic methods in applied mathematics*, pages 3–13, 1982.

6 Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1-3):3–16, 2004. doi:10.1016/S0166-218X(02)00290-1.

7 Vladimir E. Alekseev. An upper bound for the number of maximal independent sets in a graph. *Discrete Mathematics and Applications*, 17(4):355–359, 2007. doi:doi:10.1515/dma.2007.030.

8 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.

9 Ismail R. Alkhouri, George K. Atia, and Alvaro Velasquez. A differentiable approach to the maximum independent set problem using dataless neural networks. *Neural Networks*, 155:168–176, 2022.

10 J. Gary Augustson and Jack Minker. An analysis of some graph theoretical cluster techniques. *Journal of the ACM (JACM)*, 17(4):571–588, 1970.

11 Marthe Bonamy, Édouard Bonnet, Hugues Déprés, Louis Esperet, Colin Geniet, Claire Hilaire, Stéphan Thomassé, and Alexandra Wesolek. Sparse graphs with bounded induced cycle packing number have logarithmic treewidth. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3006–3028. SIAM, 2023.

12 Édouard Bonnet, Nicolas Bousquet, Pierre Charbit, Stéphan Thomassé, and Rémi Watrigant. Parameterized complexity of independent set in H-free graphs. In Christophe Paul and Michał Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation, IPEC 2018, August 20-24, 2018, Helsinki, Finland*, volume 115 of *LIPIcs*, pages 17:1–17:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.IPEC.2018.17.

13 Édouard Bonnet, Nicolas Bousquet, Stéphan Thomassé, and Rémi Watrigant. When maximum stable set can be solved in FPT time. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019, December 8-11, 2019, Shanghai University of Finance and Economics, Shanghai, China*, volume 149 of *LIPIcs*, pages 49:1–49:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.49.

14 Vincent Bouchitté and Ioan Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, 2001. doi:10.1137/S0097539799359683.

15 Andreas Brandstädt and Raffaele Mosca. Maximum weight independent set for $\ell$-claw-free graphs in polynomial time. *Discrete Applied Mathematics*, 237:57–64, 2018. doi:10.1016/j.dam.2017.11.029.

16 Sergiy Butenko. *Maximum independent set and related problems, with applications*. University of Florida, 2003.

17 Sergiy Butenko, Panos M. Pardalos, Ivan Sergienko, Vladimir Shylo, and Petro Stetsyuk. Finding maximum independent sets in graphs arising from coding theory. In Gary B. Lamont, Hisham Haddad, George A. Papadopoulos, and Brajendra Panda, editors, *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC), March 10-14, 2002, Madrid, Spain*, pages 542–546. ACM, 2002. doi:10.1145/508791.508897.

18 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

19 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. On the maximum weight independent set problem in graphs without induced cycles of length at least five. *SIAM J. Discret. Math.*, 34(2):1472–1483, 2020. doi:10.1137/19M1249473.

20 Maria Chudnovsky, Marcin Pilipczuk, Michał Pilipczuk, and Stéphan Thomassé. Quasi-polynomial time approximation schemes for the maximum weight independent set problem in $H$-free graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2260–2278, 2020. doi:10.1137/1.9781611975994.139.

21 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

22 Konrad K. Dabrowski, Vadim V. Lozin, Haiko Müller, and Dieter Rautenbach. Parameterized complexity of the weighted independent set problem beyond graphs of bounded clique number. *J. Discrete Algorithms*, 14:207–213, 2012. doi:10.1016/j.jda.2011.12.012.

23 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. III. tree-independence number of graphs with a forbidden structure. *CoRR*, abs/2206.15092, 2022. doi:10.48550/arXiv.2206.15092.

24 Clément Dallard, Martin Milanič, and Kenny Štorgel. Treewidth versus clique number. II. tree-independence number, 2021. doi:10.48550/arXiv.2111.04543.

25 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.

26 Duncan Eddy and Mykel J. Kochenderfer. A maximum independent set method for scheduling earth-observing satellite constellations. *Journal of Spacecraft and Rockets*, 58(5):1416–1429, 2021.

27 Elaine Forsyth and Leo Katz. A matrix approach to the analysis of sociometric data: preliminary report. *Sociometry*, 9(4):340–347, 1946.

28 Eleanor J. Gardiner, Peter Willett, and Peter J. Artymiuk. Graph-theoretic techniques for macromolecular docking. *J. Chem. Inf. Comput. Sci.*, 40(2):273–279, 2000. doi:10.1021/ci990262o.

29 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

30 Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

31 Peter Gartland and Daniel Lokshtanov. Independent set on $P_k$-free graphs in quasi-polynomial time. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 613–624. IEEE, 2020. doi:10.1109/FOCS46700.2020.00063.

**32** Peter Gartland, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Finding large induced sparse subgraphs in $C_{>t}$ -free graphs in quasipolynomial time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 330–341. ACM, 2021. doi:10.1145/3406325.3451034.

**33** Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

**34** Andrzej Grzesik, Tereza Klimosová, Marcin Pilipczuk, and Michał Pilipczuk. Polynomial-time algorithm for maximum weight independent set on $P_6$-free graphs. *ACM Trans. Algorithms*, 18(1):4:1–4:57, 2022. doi:10.1145/3414473.

**35** Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. In *Acta Mathematica*, pages 627–636, 1996.

**36** Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

**37** Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

**38** Denés Kőnig. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

**39** Tuukka Korhonen. Grid induced minor theorem for graphs of small degree. *J. Comb. Theory, Ser. B*, 160:206–214, 2023. doi:10.1016/j.jctb.2023.01.002.

**40** Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017. doi:10.1007/s10732-017-9337-x.

**41** Vadim V. Lozin and Martin Milanič. A polynomial algorithm to find an independent set of maximum weight in a fork-free graph. *J. Discrete Algorithms*, 6(4):595–604, 2008. doi:10.1016/j.jda.2008.04.001.

**42** Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for independent set in $P_t$-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

**43** Svatopluk Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15(2):307–309, 1974.

**44** Thomas Pontoizeau, Florian Sikora, Florian Yger, and Tristan Cazenave. Neural maximum independent set. In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases – International Workshops of ECML PKDD 2021, Virtual Event, September 13-17, 2021, Proceedings, Part I*, volume 1524 of *Communications in Computer and Information Science*, pages 223–237. Springer, 2021. doi:10.1007/978-3-030-93736-2_18.

**45** Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

**46** Donald J. Rose, Robert E. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

**47** Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM J. Comput.*, 6(3):505–517, 1977. doi:10.1137/0206036.

**48** Bram Verweij and Karen Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In Jaroslav Nesetril, editor, *Algorithms – ESA '99, 7th Annual European Symposium, Prague, Czech Republic, July 16-18, 1999, Proceedings*, volume 1643 of *Lecture Notes in Computer Science*, pages 426–437. Springer, 1999. doi:10.1007/3-540-48481-7_37.

**49** Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Inf. Comput.*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

**50** David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. doi:10.4086/toc.2007.v003a006.

# Faster 0-1-Knapsack via Near-Convex Min-Plus-Convolution

**Karl Bringmann**
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

**Alejandro Cassis**
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

## Abstract

We revisit the classic 0-1-Knapsack problem, in which we are given $n$ items with their weights and profits as well as a weight budget $W$, and the goal is to find a subset of items of total weight at most $W$ that maximizes the total profit. We study pseudopolynomial-time algorithms parameterized by the largest profit of any item $p_{\max}$, and the largest weight of any item $w_{\max}$. Our main result are algorithms for 0-1-Knapsack running in time $\widetilde{O}(nw_{\max}p_{\max}^{2/3})$ and $\widetilde{O}(np_{\max}w_{\max}^{2/3})$, improving upon an algorithm in time $O(np_{\max}w_{\max})$ by Pisinger [J. Algorithms '99]. In the regime $p_{\max} \approx w_{\max} \approx n$ (and $W \approx \mathrm{OPT} \approx n^2$) our algorithms are the first to break the cubic barrier $n^3$.

To obtain our result, we give an efficient algorithm to compute the min-plus convolution of *near-convex* functions. More precisely, we say that a function $f\colon [n] \mapsto \mathbf{Z}$ is $\Delta$-near convex with $\Delta \geq 1$, if there is a convex function $\breve{f}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$ for every $i$. We design an algorithm computing the min-plus convolution of two $\Delta$-near convex functions in time $\widetilde{O}(n\Delta)$. This tool can replace the usage of the *prediction technique* of Bateni, Hajiaghayi, Seddighin and Stein [STOC '18] in all applications we are aware of, and we believe it has wider applicability.

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

**Keywords and phrases** Knapsack, Fine-Grained Complexity, Min-Plus Convolution

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.24

**Related Version** *Full Version*: https://arxiv.org/abs/2305.01593

**Funding** This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

## 1 Introduction

In the 0-1-Knapsack problem, we are given a set of $n$ items $\mathcal{I} = \{ (p_1, w_1), \dots, (p_n, w_n) \}$, where item $i$ has a profit $p_i \in \mathbf{N}$ and a weight $w_i \in \mathbf{N}$, as well as a weight budget $W \in \mathbf{N}$. The goal is to compute $\mathrm{OPT} := \max \sum_{i=1}^n p_i x_i$ subject to the contraints $\sum_{i=1}^n w_i x_i \leq W$ and $x \in \{0,1\}^n$. This classic and fundamental problem in computer science and operations research has been studied for decades (see e.g. [29] for a book on the topic and related problems). Knapsack is weakly NP-hard, and the textbook dynamic programming algorithm due to Bellman [5] solves it in time $O(n \cdot \min\{W, \mathrm{OPT}\})$.

Recent works have studied the fine-grained complexity of Knapsack and related problems, where the goal is to give best-possible pseudopolynomial-time algorithms with respect to different parameters, see Table 1 and [7, 27, 12, 2, 15, 22, 30, 23, 26]. In this work we study the complexity of 0-1-Knapsack in terms of two natural parameters: the largest weight among the items denoted by $w_{\max}$, and the largest profit denoted by $p_{\max}$. Note that we can assume

without loss of generality that $w_{\max} \leq W$ and $p_{\max} \leq \mathrm{OPT}$. Therefore, a small polynomial dependence on these parameters can lead to faster algorithms compared to the standard dynamic programming algorithm on certain instances.

This parameterization has been studied by several previous works, see Table 1. To compare these running times, note that since any feasible solution includes at most all items, we can assume without loss of generality that $W \leq nw_{\max}$ and $\mathrm{OPT} \leq np_{\max}$. Note that when $p_{\max} \approx w_{\max} \approx n$ (and $W \approx \mathrm{OPT} \approx n^2$), all known algorithms require time $\Omega(n^3)$. In particular, in this regime the algorithm in time $O(nw_{\max}p_{\max})$ of Pisinger from '99 [34] is still the best known. In this paper we *overcome this cubic barrier*:

▶ **Theorem 1.** *There is a randomized algorithm for 0-1-Knapsack that runs in time*[1] $\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max})$ *and succeeds with high probability. Using the bound* $W \leq nw_{\max}$*, this running time is at most* $\widetilde{O}(n\, w_{\max}\, p_{\max}^{2/3})$.

Symmetrically, we obtain the following:

▶ **Theorem 2.** *There is a randomized algorithm for 0-1-Knapsack that runs in time* $\widetilde{O}((w_{\max}\mathrm{OPT})^{2/3}(np_{\max})^{1/3} + np_{\max})$ *and succeeds with high probability. Using the bound* $\mathrm{OPT} \leq np_{\max}$*, this running time is at most* $\widetilde{O}(n\, p_{\max}\, w_{\max}^{2/3})$.

**Table 1** Pseudopolynomial-time algorithms for 0-1 Knapsack.

| Reference | Running Time |
|---|---|
| Bellman [5] | $O(n \cdot \min\{W, \mathrm{OPT}\})$ |
| Pisinger [34] | $O(n \cdot p_{\max} \cdot w_{\max})$ |
| Kellerer and Pferschy [28], also [4, 3] | $\widetilde{O}(n + w_{\max} \cdot W)$ |
| Bateni, Hajiaghayi, Seddighin and Stein [4] | $\widetilde{O}(n + p_{\max} \cdot W)$ |
| Axiotis and Tzamos [3] | $\widetilde{O}(n \cdot \min\{\, w_{\max}^2, p_{\max}^2 \,\})$ |
| Bateni, Hajiaghayi, Seddighin and Stein [4] | $\widetilde{O}((n+W) \cdot \min\{\, w_{\max}, p_{\max} \,\})$ |
| Polak, Rohwedder and Węgrzycki [35] | $O(n + \min\{\, w_{\max}^3, p_{\max}^3 \,\})$ |
| Bringmann and Cassis [8] | $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$ |
| **Theorem 1** | $\widetilde{O}(n \cdot w_{\max} \cdot p_{\max}^{2/3})$ |
| **Theorem 2** | $\widetilde{O}(n \cdot p_{\max} \cdot w_{\max}^{2/3})$ |

**Min-Plus Convolution.** Given functions $f, g\colon [\,n\,] \mapsto \mathbf{Z}$, their min-plus convolution is the function $h\colon [\,2n\,] \mapsto \mathbf{Z}$ defined as $h(x) = \min_{x'} f(x') + g(x - x')$ for $x \in [\,2n\,]$. This can be trivially computed in time $O(n^2)$, and the best known algorithm for it runs in time $n^2/2^{\Omega(\sqrt{\log n})}$ [6, 36, 17]. The lack of faster algorithms has led to the *Min-Plus Convolution Hypothesis*, which postulates that there is no truly subquadratic algorithm for this problem [20, 32]. Despite this hypothesis, there are structured instances of min-plus convolution that can be solved faster [1, 4, 13, 16, 18]. These improvements have been *key to obtain the Knapsack algorithms listed in Table 1* (the only exception being Bellman's and Pisinger's algorithms [5, 34]):

[1] We use $\widetilde{O}(\cdot)$ to supress polylogarithmic factors in the input size and the largest input number.

- When one of the functions is convex, their min-plus convolution can be computed in time $O(n)$ using the SMAWK algorithm [1]. This has been used for Knapsack indirectly[2] by Kellerer and Pferschy [28], and explicitly by Axiotis and Tzamos [3] and Polak, Rohwedder and Węgrzycki [35].
- When the functions are monotone and have bounded entries, their min-plus convolution can be computed in time $\widetilde{O}(n^{1.5})$ by an algorithm due to Chi, Duan, Xie and Zhang [18]. This has been used for Knapsack by Bringmann and Cassis [8].
- Bateni, Hajiaghayi, Seddighin and Stein [4] introduced the *prediction technique* to show that the min-plus convolution of certain instances arising from Knapsack can be computed efficiently. More precisely, let $h$ be the min-plus convolution of two given functions $f, g\colon [n] \mapsto \mathbf{Z}$. They show that if one is given $n$ intervals $[x_i \mathinner{..} y_i]$ for $i \in [n]$ satisfying (i) $|h(i+j) - (f(i)+g(j))| \leq \Delta$ for every $i \in [n]$ and $j \in [x_i \mathinner{..} y_i]$, (ii) for every output $h(k)$ there exists at least one $i$ such that $f(i) + g(k-i) = h(k)$ and $k - i \in [x_i \mathinner{..} y_i]$ and (iii) $0 \leq x_i, y_i < n$ for all intervals and $x_i \leq x_j, y_i \leq y_j$ for all $i < j$; then $h$ can be computed in time $\widetilde{O}(n \cdot \Delta)$. They showed that this is applicable in the context of Knapsack.

Our Theorems 1 and 2 fall into the same category of improvements, as we design an efficient algorithm for a new class of structured instances of min-plus convolution, namely *near convex* functions: We say that $f\colon [n] \mapsto \mathbf{Z}$ is $\Delta$-near convex, if there is a convex function $\breve{f}\colon [n] \mapsto \mathbf{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$ for all $i \in [n]$. Our theorem reads as follows:

▶ **Theorem 3** (Near Convex MinPlus Convolution). *Let $f\colon [n] \mapsto [-U \mathinner{..} U]$, and $g\colon [m] \mapsto [-U \mathinner{..} U]$ be given as inputs where $n, m, U \in \mathbf{N}$. Let $\Delta \geq 1$ such that both $f$ and $g$ are $\Delta$-near convex. Then the min-plus convolution of $f$ and $g$ can be computed in time $\widetilde{O}((n+m) \cdot \Delta)$.*

We view our Theorem 3 as a replacement for the prediction technique by Bateni et al. [4]. Indeed, all uses of the prediction technique exploit near-convexity to ensure its preconditions, and thus all uses that we are aware of can be replaced by our Theorem 3. Since the prediction technique is both difficult to state and difficult to apply, we view our Theorem 3 as replacing the prediction technique by an *easily applicable tool with a concise statement*. Moreover, Theorem 3 provides a new tool for structured instances of min-plus convolution, which we use in this paper to make progress on 0-1-Knapsack, and which we believe has wider applicability.

**Our Techniques.** Our approach to prove Theorem 3 is as follows. Let $f, g\colon [n] \mapsto \mathbf{Z}$ be the input functions, and let $h$ be their min-plus convolution, which we aim to compute. First we observe that we can obtain the convex approximations $\breve{f}, \breve{g}$ witnessing the $\Delta$-near convexity of $f$ and $g$, and compute their min-plus convolution $\breve{h}$ efficiently. By exploiting $\breve{h}$ and the convexity of $\breve{f}$ and $\breve{g}$, we identify a structured set $R \subseteq [n]^2$ with the property that any $(i,j) \in [n]^2 \setminus R$ satisfies $f(i) + g(j) > h(j)$. Then, we give a simple recursive algorithm to cover $R$ with a collection $\mathcal{C}$ of disjoint dyadic boxes $I \times J$ , where $(I, J) \in \mathcal{C}$ satisfies $I, J \subseteq [n]$ and $I \times J \subseteq R$. Thus, we can infer $h$ by computing the *sumset* $A := \{(i, f(i)) \mid i \in I\} + \{(j, g(j)) \mid j \in J\}$ and taking $h(k) = \min\{y \mid (k, y) \in A\}$ for every $(I, J) \in \mathcal{C}$. To do this efficiently we observe that inside $I$ and $J$, the functions $f[I]$ and $g[J]$ are close to linear functions with the same slope up to an additive error of $\pm O(\Delta)$ (which follows from their $\Delta$-near convexity). This implies that their sumset is small; more

[2] Kellerer and Pferschy did not use SMAWK, but gave a different algorithm for computing the min-plus convolution of these instances in time $O(n \log n)$.

precisely it has size $O((|I|+|J|)\Delta)$. Finally, we make use of known tools that can compute a sumset in time proportional to its size. The idea of identifying a covering with small sumsets to efficiently compute the min-plus convolution is inspired by Chan and Lewenstein's [16] algorithm for bounded monotone sequences (in which they do not use convexity in any form). Our algorithm shares some similarities with the prediction technique by Bateni et al. [4]. In particular, the covering by dyadic boxes where functions are near-linear resembles the way in which they exploit the intervals $[\,x_i\,..\,y_i\,]$ required by their algorithm.

To obtain Theorems 1 and 2, we follow the *partition and convolve* paradigm that has been used in many recent algorithms for Subset Sum and Knapsack, see e.g. [7, 4, 8, 26, 14, 31, 21, 11]. Specifically, we randomly split the items into $q$ groups. In each group, we use the standard dynamic programming algorithm to compute for each weight $i$, the maximum profit $f(i)$ attainable with weight at most $i$ using items from that group. Then we combine the functions $f$ over all groups by min-plus convolution. The crucial observation is that due to the random splitting we only need to compute the values $f(i)$ in a small weight interval.

**Further Related Work.** Cygan et al. [20] and Künnemann et al. [32] showed that under the Min-Plus Convolution Hypothesis, there is no truly subquadratic algorithm for Knapsack on instances with $w_{\max}, W = \Theta(n)$ and $p_{\max}, \mathrm{OPT} = \Omega(n^2)$, and symmetrically, on instances with $p_{\max}, \mathrm{OPT} = \Theta(n)$ and $w_{\max}, W = \Omega(n^2)$. This implies that Bellman's dynamic programming algorithm is conditionally optimal in these settings.

Pseudopolynomial-time algorithms parameterized by $p_{\max}$ and $w_{\max}$ have also been studied for the closely related Unbounded Knapsack problem. Here, the setup is the same as for 0-1 Knapsack but now a solution might include an arbitrary number of copies of each item. Chan and He [15] gave an algorithm for this problem in time $\widetilde{O}(n \cdot \min\{\, p_{\max}, w_{\max}\,\})$, which is optimal under the Min-Plus Convolution Hypothesis. Bringmann and Cassis [8] gave an algorithm in time $\widetilde{O}(n + (p_{\max}+w_{\max})^{1.5})$ which is better when $w_{\max} \approx p_{\max} \approx n$.

**Outline.** The paper is organized as follows. In Section 2 we give some formal preliminaries and establish some notation. In Section 3 we give our algorithm for Knapsack proving Theorems 1 and 2, assuming Theorem 3. In Section 4 we will then give our algorithm for min-plus convolution, proving Theorem 3.

## 2 Preliminaries

We write $\mathbf{N} = \{\,0, 1, 2, \dots\,\}$. For $t \in \mathbf{N}$, we define $[\,t\,] := \{\,0, 1, \dots, t\,\}$. Let $A \in \mathbf{Z}^{n+1}$ be an integer sequence, i.e., $A[i] \in \mathbf{Z}$ for $i \in [\,n\,]$. Sometimes we will refer to such a sequence as a *function* $A\colon [\,n\,] \mapsto \mathbf{Z}$. With this in mind, we use the notation $-A$ to denote the entry-wise negation of $A$. Given $a, b \in \mathbf{R}$ with $a \le b$, we define $[\,a\,..\,b\,] := \{\,\max(0, \lfloor a \rfloor), \max(0, \lfloor a \rfloor)+1, \dots, \lceil b \rceil - 1, \lceil b \rceil\,\}$. The non-standard rounding and capping at 0 in the definition of $[\,a\,..\,b\,]$ is useful to index a subsequence $A[\,a\,..\,b\,]$ when $a$ and $b$ might not be non-negative integers.

The max-plus convolution of two sequences $A[\,0\,..\,n\,] \in \mathbf{Z}^{n+1}, B[\,0\,..\,m\,] \in \mathbf{Z}^{m+1}$, denoted by $\textsc{MaxConv}(A, B)$, is a sequence of length $n+m+1$ where for each $k \in [\,n+m\,]$ we have $\textsc{MaxConv}(A,B)[k] := \max_{i+j=k} A[i] + B[j]$. The min-plus convolution $\textsc{MinConv}(A, B)$ is defined analogously, but replacing max by a min. Note that by negating the entries of the sequences, these two operations are equivalent.

▶ **Fact 4.** *For any $A \in \mathbf{Z}^{n+1}, B \in \mathbf{Z}^{m+1}$, we have $\textsc{MaxConv}(A,B) = -\textsc{MinConv}(-A,-B)$.*

We will use the following handy notation: Given sequences $A[0\,..\,n], B[0\,..\,n]$ and intervals $I, J \subseteq [n]$ and $K \subseteq [2n]$, we denote by $C[K] := \textsc{MaxConv}(A[I], B[J])$ the computation of $C[k] := \max\{A[i] + B[j] \colon i \in I, j \in J, i + j = k\}$ for each $k \in K$.

We say that a function $f \colon [n] \mapsto \mathbf{Q}$ is *convex* if $f(i) - f(i-1) \leq f(i+1) - f(i)$ holds for every $i \in [1\,..\,n-1]$. We say that $f$ is *concave* if $-f$ is convex.

▶ **Definition 5** (Near Convex and Near Concave Functions). *For $\Delta \geq 0$, we say that a function $f \colon [n] \mapsto \mathbf{Z}$ is $\Delta$-near convex, if there is a convex function $\breve{f} \colon [n] \mapsto \mathbf{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$. We say that $f$ is $\Delta$-near concave if $-f$ is $\Delta$-near convex.*

If the input consists of $N$ numbers in $[-U\,..\,U]$, we denote $\widetilde{O}(T) = \bigcup_{c \geq 0} O(T \log^c(NU))$.

## 3 Faster 0-1 Knapsack Algorithm

In this section we prove Theorem 1. Let $(\mathcal{I}, W)$ be a 0-1 Knapsack instance. Throughout, we denote the number of items by $n := |\mathcal{I}|$. We identify the item set $\mathcal{I}$ with $\{1, \ldots, n\}$. We represent a *solution* to the knapsack instance (i.e., a subset of $\mathcal{I}$), by an indicator vector $x \in \{0,1\}^n$. For a subset of the items $\mathcal{J} \subseteq \mathcal{I}$, we put $w_{\mathcal{J}}(x) := \sum_{i \in \mathcal{J}} w_i x_i$ and $p_{\mathcal{J}}(x) := \sum_{i \in \mathcal{J}} p_i x_i$. We define the profit sequence $\mathcal{P}_{\mathcal{I}}[\cdot]$, where for each $j \in \mathbf{N}$ we have

$$\mathcal{P}_{\mathcal{I}}[j] = \max\{\, p_{\mathcal{I}}(x) \mid x \in \{0,1\}^n, w_{\mathcal{I}}(x) \leq j \,\}.$$

Observe that $\mathcal{P}_{\mathcal{I}}$ is monotone non-decreasing, and that $\mathrm{OPT} = \mathcal{P}_{\mathcal{I}}[W]$. The textbook way to compute $\mathcal{P}_{\mathcal{I}}[0\,..\,j]$ is to use dynamic programming:

▶ **Fact 6.** *For any $j \in \mathbf{N}$ the sequence $\mathcal{P}_{\mathcal{I}}[0\,..\,j]$ can be computed in time $O(nj)$.*

Before presenting the algorithm, we make two simple observations about the given Knapsack instance $(\mathcal{I}, W)$. First, by ignoring items with weight larger than the capacity $W$, we can assume without loss of generality that $w_{\max} \leq W$. Now every single item is a feasible solution, so we have $p_{\max} \leq \mathrm{OPT}$. Second, observe that if $W \geq n \cdot w_{\max}$, then the instance is trivial since we can pack all items. Thus, we can assume without loss of generality that $W \leq n \cdot w_{\max}$. Moreover, since any feasible solution consists of at most all the $n$ items, it follows that $\mathrm{OPT} \leq n \cdot p_{\max}$.

### The Algorithm

We now describe the algorithm. Set parameters $q := \min\{(n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\}$ rounded down to the closest power of 2, $\Delta := w_{\max} W/q$ and $\eta := 11 \log n$. For each $\ell \in [\log q]$ we define the interval $J^\ell := [\frac{W}{q} 2^\ell - \sqrt{\Delta 2^\ell}\eta\,..\,\frac{W}{q} 2^\ell + \sqrt{\Delta 2^\ell}\eta]$.

We start by splitting the items $\mathcal{I}$ into $q$ groups $\mathcal{I}_1^0, \ldots, \mathcal{I}_q^0$ uniformly at random. The idea will be to compute an array $C_j^0$ associated to each $\mathcal{I}_j^0$, and then combine them in a *tree-like* fashion. A crucial aspect for the running time is that we only compute $|J^\ell|$ entries of each array $C_j^\ell$. In detail, we proceed as follows:

**Base Case.** For each $\mathcal{I}_j^0$, we use Fact 6 to compute $\mathcal{P}_{\mathcal{I}_j^0}[0\,..\,\frac{W}{q} + \sqrt{\Delta}\eta]$ and define the subarray $C_j^0[J^0] := \mathcal{P}_{\mathcal{I}_j^0}[J^0]$.

**Combination.** Iterate over the *levels* $\ell = 1, \ldots \log(q)$. For $j \in [1\,..\,q/2^\ell]$ we set $\mathcal{I}_j^\ell := \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$. Then, compute the subarray $C_j^\ell[J^\ell]$ by taking the relevant entries of the max-plus convolution of $C_{2j-1}^{\ell-1}[J^{\ell-1}]$ and $C_{2j}^{\ell-1}[J^{\ell-1}]$.

**Returning the answer.** (Note that when $\ell = \log(q)$, it holds that $\mathcal{I}_1^{\log q} = \mathcal{I}$.) We return the value $C_1^{\log q}[W]$. See Algorithm 1 for the pseudocode.

**Algorithm 1** Knapsack Algorithm. Given a set of items $\mathcal{I}$ and a weight budget $W$, the algorithm computes the maximum attainable profit.

$q \leftarrow \min\{ (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max} \}$ rounded down to the closest power of 2
$\Delta \leftarrow w_{\max}W/q$
$\eta \leftarrow 11 \log n$
$\mathcal{I}_1^0, \dots, \mathcal{I}_q^0 \leftarrow$ random partitioning of $\mathcal{I}$ into $q$ groups
**for** $i = 1 \dots q$ **do**
    Compute $\mathcal{P}_{\mathcal{I}_j^0}[\,0\,..\,\frac{W}{q} + \sqrt{\Delta}\eta\,]$ using standard dynamic programming (Fact 6)
    $J^0 \leftarrow [\,\frac{W}{q} - \sqrt{\Delta}\eta\,..\,\frac{W}{q} + \sqrt{\Delta}\eta\,]$
    $C_j^0[J^0] \leftarrow \mathcal{P}_{\mathcal{I}_j^0}[J^0]$
**for** $\ell = 1 \dots \log(q)$ **do**
    $J^\ell \leftarrow [\,\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta\,..\,\frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta\,]$
    **for** $j = 1, \dots, q/2^\ell$ **do**
        $\mathcal{I}_j^\ell \leftarrow \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$
        Compute $C_j^\ell[J^\ell] \leftarrow \textsc{MaxConv}(C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}])$ using Theorem 3
**return** $C_1^{\log q}[W]$

## Correctness

We start by analyzing the correctness of the algorithm. The following lemma shows that the weight of any solution restricted to one of the sets $\mathcal{I}_j^\ell$ is concentrated around its expectation.

▶ **Lemma 7** (Concentration, proof deferred to the full version). *Let $x \in \{0,1\}^n$ be a solution to the given Knapsack instance. Fix a level $\ell \in [\,0\,..\,\log q\,]$ and $j \in [\,1\,..\,q/2^\ell\,]$. Then, with probability at least $1 - 1/n^4$ it holds that:*

$$\left| w_{\mathcal{I}_j^\ell}(x) - \frac{w_{\mathcal{I}}(x) \cdot 2^\ell}{q} \right| \leq \sqrt{\Delta 2^\ell} \cdot 10 \log n.$$

Using Lemma 7, we can argue that at level $\ell$ it suffices to compute a subarray of length $\widetilde{O}(\sqrt{\Delta 2^\ell})$ around $W2^\ell/q$. The following lemma makes this precise:

▶ **Lemma 8** (Proof deferred to the full version). *Let $x \in \{0,1\}^n$ be a solution to the given Knapsack instance satisfying $w_{\mathcal{I}}(x) \in [\,W - w_{\max}\,..\,W\,]$. With probability at least $1 - 1/n^2$, for all levels $\ell \in [\,0\,..\,\log q\,]$ and all $j \in [\,1\,..\,q/2^\ell\,]$ it holds that:*

- $w_{\mathcal{I}_j^\ell}(x) \in J^\ell = [\,\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta\,..\,\frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta\,]$, *and*
- $C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] \geq p_{\mathcal{I}_j^\ell}(x)$.

▶ **Lemma 9** (Correctness of Algorithm 1). *Let $x^* \in \{0,1\}^n$ be an optimal solution to the given Knapsack instance. Then, for every $i \in [\,w_{\mathcal{I}}(x^*)\,..\,W\,]$, it holds that $C_1^{\log q}[i] = \mathcal{P}_{\mathcal{I}}[i]$ with probability at least $1 - 1/n^2$.*

**Proof.** We can check in linear time $O(n)$ whether the optimal solution consists of all items, in which case the instance is trivial. Thus, we can assume without loss of generality that $x^*$ does not include all items. In particular, $x^*$ leaves at least one item out and therefore its weight satisfies $w_{\mathcal{I}}(x^*) \in [\,W - w_{\max}\,..\,W\,]$. By Lemma 8, it holds that $C_1^{\log q}[w_{\mathcal{I}}(x^*)] \geq p_{\mathcal{I}}(x^*) = \mathcal{P}_{\mathcal{I}}[w_{\mathcal{I}}(x^*)]$ with probability at least $1 - 1/n^2$. From now on we condition on this event. We will use the following auxiliary claim:

▷ Claim 10. The sequence $C_1^{\log q}[J^{\log q}]$ is monotone non-decreasing, and satisfies $C_1^{\log q}[i] \leq \mathcal{P}_{\mathcal{I}}[i]$ for all $i \in J^{\log q}$.

Proof. First we argue monotonicity by induction. Note that in the base case $\ell = 0$, the sequence $C_j^0[J^0] = \mathcal{P}_{\mathcal{I}_j^0}[J^0]$ is monotone non-decreasing due to the definition of $\mathcal{P}_{\mathcal{I}_j^0}$. For level $\ell > 0$, the sequence $C_j^\ell$ is computed by taking the max-plus convolution of sequences of level $\ell - 1$. The result follows by observing that the max-plus convolution of two monotone non-decreasing sequences is monotone non-decreasing.

The second part of the claim follows since (inductively) every entry $C_1^{\log q}[i]$ for $i \in J^{\log q}$ corresponds to the profit of a subset of items of $\mathcal{I}$ of weight at most $i$. ◁

Since $x^*$ is an optimal solution, it holds that $\mathcal{P}_{\mathcal{I}}[i] = p_{\mathcal{I}}(x^*)$ for all $i \in [\, w_{\mathcal{I}}(x^*) \,..\, W\,]$. Thus Claim 10 yields that $C_1^{\log q}[i] = \mathcal{P}_{\mathcal{I}}[i]$ for all $i \in [\, w_{\mathcal{I}}(x^*) \,..\, W\,]$, completing the proof. ◀

## Running Time

Now we analyze the running time of Algorithm 1. The key speedup comes from the computation in Algorithm 1, where we use Theorem 3 to perform the max-plus convolution. Since Theorem 3 is phrased in terms of min-plus convolution of near-convex functions, we will use the following corollary:

▶ **Corollary 11.** *Let $f\colon [\,n\,] \mapsto [\,-U \,..\, U\,]$ and $g\colon [\,m\,] \mapsto [\,-U \,..\, U\,]$ be given as inputs, where $U \in \mathbf{N}$. Let $\Delta \geq 1$ be such that both $f$ and $g$ are $\Delta$-near concave. Then, $\textsc{MaxConv}(f, g)$ can be computed in time $\widetilde{O}((n+m)\Delta)$*

**Proof.** Noting that $-f$ and $-g$ are $\Delta$-near convex (Definition 5), the result follows from Theorem 3 and Fact 4. ◀

▶ **Lemma 12** (Near Concavity, proof deferred to the full version)**.** *For every level $\ell \in [\,1 \,..\, q\,]$ and every $j \in [\,1 \,..\, q/2^\ell\,]$, it holds that $C_j^\ell[J^\ell]$ is $p_{\max}$-near concave.*

▶ **Lemma 13.** *Fix a level $\ell \in [\,1 \,..\, q\,]$ and an iteration $j \in [\,1 \,..\, q/2^\ell\,]$. The computation of $C_j^\ell$ in Algorithm 1 takes time $\widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$*

**Proof.** By Lemma 12, the sequences $C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}]$ are $p_{\max}$-near concave. Thus, by Corollary 11, their max-plus convolution can be computed in time $\widetilde{O}(p_{\max}|J^\ell|) = \widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$, where we used $\eta = \widetilde{O}(1)$. ◀

▶ **Lemma 14** (Running Time of Algorithm 1)**.** *Algorithm 1 runs in time*

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max}).$$

**Proof.** Recall that $q = \min\{\, (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\}$ (up to a factor of 2). Since $W \leq nw_{\max}$, we have that $q \leq n$. Moreover, since we assume without loss of generality that $w_{\max} \leq n$, note that $q < 1$ if and only if $q = (n/p_{\max})^{2/3}(W/w_{\max})^{1/3} < 1$. This implies that $p_{\max} > n\sqrt{W/w_{\max}}$. But in this case, the claimed running time is $\Omega(nW)$, so the standard $O(nW)$ dynamic programming algorithm (Fact 6) already achieves our time bound. Thus, we can assume without loss of generality that $1 \leq q \leq n$, i.e., $q$ is a valid choice for the number of groups in which we split the item set $\mathcal{I}$.

We start bounding the running time of the base case, i.e., the computation of the arrays $C_j^0$ for $j \in [\,1\,..\,q\,]$ in Algorithm 1. By Fact 6, and the definition $\Delta = w_{\max}W/q$ this takes time

$$O\left(\sum_{j=1}^{q} |\mathcal{I}_j^0|(\tfrac{W}{q} + \sqrt{\Delta}\eta)\right) = O\left(n(\tfrac{W}{q} + \sqrt{\Delta}\eta)\right) = \widetilde{O}\left(n\tfrac{W}{q} + n\sqrt{\tfrac{w_{\max}W}{q}}\right). \tag{1}$$

Now we bound the time of the combination step done in Algorithms 1–1. At level $\ell \in [\,1\,..\,q\,]$ and iteration $j \in [\,1\,..\,q/2^\ell\,]$ the execution of Algorithm 1 takes time $\widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$ by Lemma 13. Thus, we can bound the overall time as

$$\sum_{\ell=1}^{\log q}\sum_{j=1}^{q/2^\ell} \widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell}) = \sum_{\ell=1}^{\log q} \frac{q}{2^\ell}\widetilde{O}\left(p_{\max}\sqrt{\tfrac{w_{\max}W}{q}2^\ell}\right) = \sum_{\ell=1}^{\log q} \widetilde{O}\left(p_{\max}\sqrt{\tfrac{qw_{\max}W}{2^\ell}}\right),$$

since this is a geometric series, it is bounded by the first term $\widetilde{O}(p_{\max}\sqrt{qw_{\max}W})$. Combining this with (1), we obtain overall time

$$\widetilde{O}\left(p_{\max}\sqrt{qw_{\max}W} + n\tfrac{W}{q} + n\sqrt{\tfrac{w_{\max}W}{q}}\right).$$

Recalling that $q = \Theta(\min\{\,(n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\})$, we obtain overall time

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max} + (p_{\max}W)^{1/3}(nw_{\max})^{2/3}).$$

Finally, using that $\sqrt{xy} \le (x+y)/2$ for all $x, y \ge 0$, we have that

$$\begin{aligned}(p_{\max}W)^{1/3}(nw_{\max})^{2/3} &= \sqrt{(p_{\max}W)^{2/3}(nw_{\max})^{1/3}nw_{\max}} \\ &\le O((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max}).\end{aligned}$$

Thus, the overall running time is $\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max})$, as claimed. ◀

**Proof of Theorem 1.** Run Algorithm 1. By Lemma 9, we obtain that $\mathcal{I}_1^{\log q}[W] = \mathrm{OPT}$ with probability at least $1 - 1/n^2$, which proves correctness. The running time is immediate from Lemma 14. Observe that we can obtain success probability $1 - 1/n^c$ for any constant $c \ge 2$ by repeating the algorithm $c/2$ times. Finally, note that Algorithm 1 only computes the optimal profit of the given instance. In the full version of the paper we describe how to reconstruct the set of items in an optimal solution with no overhead in the running time. ◀

**Proof Sketch of Theorem 2.** Our presentation focused on proving Theorem 1. The proof of the symmetric variant stated in Theorem 2 is very similar, thus we only sketch the required changes. Essentially, we need to exchange profits with weights everywhere, which in turn means exchanging max-plus convolutions by min-plus convolutions. In more detail: Instead of working with the profit sequence $\mathcal{P}_\mathcal{I}$, we work with the weight sequence $\mathcal{W}_\mathcal{I}$, where the entry $\mathcal{W}_\mathcal{I}[j]$ stores the minimum weight of a solution with profit at least $j$. We do not know OPT, but we can compute an approximation $\tilde{V}$ satisfying $\tilde{V} - p_{\max} \le \mathrm{OPT} \le \tilde{V}$ in linear time (see e.g. [29, Theorem 2.5.4]). In the algorithm, we exchange all ocurrences of $w_{\max}$ by $p_{\max}$ and all ocurrences of $W$ by $\tilde{V}$. With these changes, the functions $C_j^\ell$ are now $w_{\max}$-near convex (instead of $p_{\max}$-near concave) so we use Theorem 3 directly instead of Corollary 11. In this way, we obtain the array $C_1^{\log q}[\,\tilde{V} - p_{\max}\,..\,\tilde{V}\,] = \mathcal{W}_\mathcal{I}[\,\tilde{V} - p_{\max}\,..\,\tilde{V}\,]$. Then, we can infer OPT as the largest $i \in [\,\tilde{V} - p_{\max}\,..\,\tilde{V}\,]$ such that $\mathcal{W}_\mathcal{I}[i] \le W$. ◀

# 4 MinPlus Convolution for Near-Convex Sequences

In this section we prove Theorem 3.

▶ **Theorem 3** (Near Convex MinPlus Convolution). *Let $f\colon [n] \mapsto [-U\,..\,U]$, and $g\colon [m] \mapsto [-U\,..\,U]$ be given as inputs where $n, m, U \in \mathbf{N}$. Let $\Delta \geq 1$ such that both $f$ and $g$ are $\Delta$-near convex. Then the min-plus convolution of $f$ and $g$ can be computed in time $\widetilde{O}((n+m)\cdot\Delta)$.*

## 4.1 Preparations

Throughout this section, fix the functions $f\colon [n] \mapsto [-U\,..\,U], g\colon [m] \mapsto [-U\,..\,U]$. Recall that we say that $f\colon [n] \mapsto \mathbf{Z}$ is $\Delta_f$-near convex, if there is a convex function $\breve{f}\colon [n] \mapsto \mathbf{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta_f$ for all $i \in [n]$ (see Definition 5). First observe that the lower convex hull of the points $\{ (i, f(i)) \mid i \in [n] \}$ gives the pointwise maximal convex function $\breve{f}$ with $\breve{f} \leq f$. This can be computed in time $O(n)$ by Graham's scan [25], since the points are already sorted by $x$-coordinate. Then, we can infer $\Delta_f = \max\{ 1, \max_{i\in[n]} f(i) - \breve{f}(i) \}$. Thus, from now on we assume that we know $\breve{f}, \Delta_f, \breve{g}, \Delta_g$. Set $\Delta := \max\{ \Delta_f, \Delta_g \}$. Let $\breve{h} := \textsc{MinConv}(\breve{f}, \breve{g})$ and $h := \textsc{MinConv}(f, g)$. The goal is to compute $h$.

We start by introducing some notation. We call $(i, j) \in [n] \times [m]$ a *point*. We visualize a point $(i, j)$ as lying on the $i$-th row and $j$-th column of an $n \times m$ grid, where $(0, 0)$ is on the bottom-left corner and $(n, m)$ on the top right corner. A point $(i, j)$ lies on *diagonal* $i + j$. For any $\delta \geq 0$, a point $(i, j)$ is *$\delta$-relevant* if $\breve{f}(i) + \breve{g}(j) \leq \breve{h}(i + j) + \delta$. We denote by $R_\delta$ the set of all $\delta$-relevant points.

Points that are 0-relevant are important because of the following observation: We call $i$ a *witness* for $\breve{h}(k)$ if $\breve{f}(i) + \breve{g}(k - i) = \breve{h}(k)$. Thus, observe that $i$ is a witness for $\breve{h}(k)$ if and only if $(i, k - i)$ is a 0-relevant point.

The importance of $2\Delta$-relevant points is captured by the following lemma:

▶ **Lemma 15.** *If $(i, j) \notin R_{2\Delta}$ then $f(i) + g(j) > h(i + j)$.*

That is, points that are not $2\Delta$-relevant can be ignored for the purpose of computing $h$.

**Proof.** Since $(i, j)$ is not $2\Delta$-relevant, it holds that $f(i) + g(j) \geq \breve{f}(i) + \breve{g}(j) > \breve{h}(i + j) + 2\Delta$. Let $k := i + j$, and let $i^*$ be a witness for $\breve{h}(k)$, i.e., $\breve{f}(i^*) + \breve{g}(k - i^*) = \breve{h}(k)$. Then,

$$h(k) \leq f(i^*) + g(k - i^*) \leq \breve{f}(i^*) + \Delta + \breve{g}(k - i^*) + \Delta = \breve{h}(k) + 2\Delta < f(i) + g(j). \qquad \blacktriangleleft$$

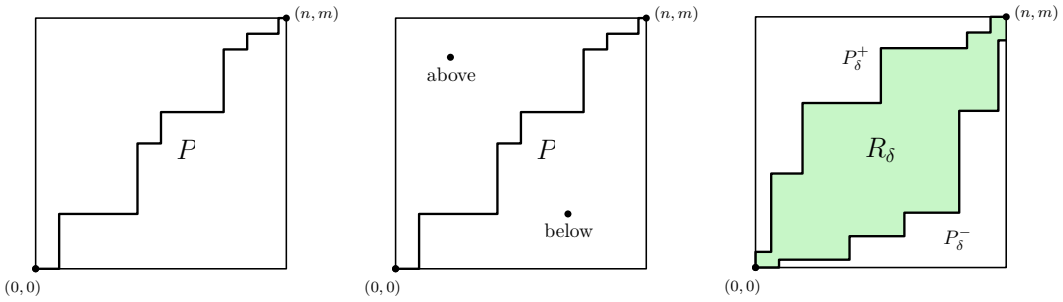
We say that a set of points $P$ is a *monotone path* if for every $k \in [n + m]$ $P$ contains exactly one point $(i_k, j_k)$ on diagonal $k$, and we have $(i_{k+1}, j_{k+1}) \in \{ (i_k + 1, j_k), (i_k, j_k + 1) \}$ for every $k \in [n + m - 1]$, see Figure 1a for an illustration. For any $\delta > 0$, we let

$$P_\delta^+ := \{ (i, k - i) \mid k \in [n + m], i \in [n] \text{ is maximal s.t. } (i, k - i) \text{ is } \delta\text{-relevant} \},$$
$$P_\delta^- := \{ (i, k - i) \mid k \in [n + m], i \in [n] \text{ is minimal s.t. } (i, k - i) \text{ is } \delta\text{-relevant} \}.$$

The next two lemmas show that $P_\delta^+, P_\delta^-$ are monotone paths and that $P_\delta^+, P_\delta^-$ form the boundary of $R_\delta$, see Figure 1c for an illustration. This establishes structure of $R_\delta$ that we will be exploit later.

▶ **Lemma 16** (Monotone Paths, proof deferred to the full version). *For any $\delta \geq 0$, $P_\delta^-, P_\delta^+$ are monotone paths.*

Let $(i, j)$ be a point and $P$ a monotone path. Let $(a, b) \in P$ be the unique point on the same diagonal as $(i, j)$, i.e., $a + b = i + j$. We say that $(i, j)$ is *below* $P$ if $i < a$, *above* $P$ if $i > a$, and *on* $P$ if $i = a$, see Figure 1b for an illustration.




**(a)** A monotone path $P$. **(b)** Points above and below $P$. **(c)** $R_\delta$ is between $P_\delta^+$ and $P_\delta^-$.

**Figure 1** Visualizations for concepts used in Section 4.

▶ **Lemma 17.** *For any $\delta \geq 0$, $R_\delta$ consists of all points $(i, j)$ that are on or below $P_\delta^+$ and on or above $P_\delta^-$.*

**Proof.** Fix $k \in [\, n + m \,]$ and let $(i^+, k - i^+), (i^-, k - i^-)$ be the point on diagonal $k$ in $P_\delta^+$ and $P_\delta^-$, respectively. Consider any $(i, j) \in R_\delta$ on diagonal $k$. By maximality of $i^+$ we have $i \leq i^+$, and similarly $i \geq i^-$ by the minimality of $i^-$. Thus, no point in $R_\delta$ is above $P_\delta^+$ or below $P_\delta^-$. It remains to show that for any $i^- \leq i \leq i^+$ we have $(i, k - i) \in R_\delta$. Note that the function $r(i) := \breve{f}(i) + \breve{g}(k - i)$ is convex (since it is the sum of convex functions). Since $(i^+, k - i^+)$ is $\delta$-relevant, we have $r(i^+) \leq \breve{h}(k) + \delta$. Similarly, since $(i^-, k - i^-)$ is $\delta$-relevant, we have $r(i^-) \leq \breve{h}(k) + \delta$. By convexity of $r$, we obtain that $r(i) \leq \breve{h}(k) + \delta$ for all $i^- \leq i \leq i^+$. Hence, we conclude that for each $i^- \leq i \leq i^+$ we have $(i, k - i) \in R_\delta$. ◀

Finally, we need some background on *sumsets*. Given $A, B \subseteq [\, -U \,..\, U \,]^2$ where $U \in \mathbf{N}$, we define $A + B = \{\, a + b \mid a \in A, b \in B \,\}$ as their sumset, where the addition $a + b$ is done componentwise. The naive way to compute $A + B$ takes time $O(|A| \cdot |B|)$. For our application, we want to compute the sumset in time near linear in its size $|A + B|$. For this end, we will use the following tool to compute *sparse non-negative convolution*. Given vectors $P, Q \in \mathbf{N}^n$, their *convolution* $P \star Q \in \mathbf{N}^{2n-1}$ is defined coordinate-wise by $(P \star Q)[k] = \sum_{i+j=k} P[i] \cdot Q[j]$.

▶ **Theorem 18** (Deterministic Sparse Convolution [10])**.** *There is a deterministic algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t \operatorname{polylog}(n\Delta))$, where $t$ is the number of non-zero entries in $A \star B$ and $\Delta$ is the largest entry in $A$ and $B$.*

See also [9] for improvements in the log-factors at the cost of randomization and [19, 33, 24] for prior randomized algorithms with similar guarantees.

▶ **Corollary 19** (Output Sensitive Sumset Computation)**.** *Given $A, B \subseteq [\, -U \,..\, U \,]^2$, with $|A + B| \leq N$, $A + B$ can be computed in time $\widetilde{O}(N)$.*

**Proof.** Let $A' := \{\, (x+U) \cdot 5U + (y+U) \mid (x, y) \in A \,\}$ and similarly, let $B' := \{\, (x+U) \cdot 5U + (y+U) \mid (x, y) \in B \,\}$. Observe that this is a one-to-one embedding of $A, B \subseteq [\, -U \,..\, U \,]^2$ into $A', B' \subseteq [\, \Theta(U^2) \,]$. Moreover, one can check that given $C' := A' + B'$ we can infer $C := A + B$ (the choice of $5U$ prevents any interactions between coordinates when summing them up).

Thus, it suffices to compute $A' + B'$. To this end, construct their indicator vectors $P_{A'}, P_{B'} \in \mathbf{N}^{\Theta(U^2)}$ and compute the convolution $P_{C'} = P_{A'} \star P_{B'}$. The non-zero entries in $P_{C'}$ correspond to the elements of $A' + B'$. By Theorem 18, this runs in time $O(|A' + B'| \operatorname{polylog}(N, U)) = \widetilde{O}(N)$. ◀

## 4.2 Algorithm

We are ready to describe our algorithm. Recall that we have access to the functions $f, \breve{f}, g, \breve{g}$ and the value $\Delta = \max\{\Delta_f, \Delta_g\}$.

**Computing $\breve{h} = \textsf{MinConv}(\breve{f}, \breve{g})$.** Consider the pseudocode given in Algorithm 2.

**Algorithm 2** Given convex functions $\breve{f}\colon [n] \mapsto \mathbf{Q}, \breve{g}\colon [m] \mapsto \mathbf{Q}$, the algorithm computes $\breve{h} = \textsc{MinConv}(\breve{f}, \breve{g})$.

1: $i_0^* \leftarrow 0, \breve{h}(0) \leftarrow \breve{f}(0) + \breve{g}(0)$
2: **for** $k = 1, \ldots, n+m$ **do**
3: $\quad i_k^* \leftarrow \operatorname{argmin}\{\breve{f}(i) + \breve{g}(k-i) + \frac{i}{2n} \mid i \in \{i_{k-1}^*, i_{k-1}^* + 1\} \cap [n]\}$
4: $\quad \breve{h}(k) \leftarrow \breve{f}(i_k^*) + \breve{g}(k - i_k^*)$

▶ **Lemma 20.** *Algorithm 2 computes $\breve{h} = \textsc{MinConv}(\breve{f}, \breve{g})$ in time $O(n+m)$.*

**Proof.** The running time is immediate. To see correctness, focus on $i_k^*$ for $k \in [n+m]$ as computed in Algorithm 2. We claim that the path $P_0^-$ equals $\{(i_k^*, k - i_k^*) \mid k \in [n+m]\}$. That is, we want to argue that $i_k^*$ is the minimum witness of $\breve{h}(k)$ for each $k \in [n+m]$. Indeed, by Lemma 16, $P_0^-$ is a monotone path. Thus, $i_k^* \in \{i_{k-1}^*, i_{k-1}^* + 1\}$. Observe that in Algorithm 2 we pick $i_k^*$ as the minimizer of $\breve{f}(i) + \breve{g}(k-i) + \frac{i}{2n}$ where $i \in \{i_{k-1}^*, i_{k-1}^* + 1\}$. Therefore, the algorithm correctly computes $i_k^*$ (the additive term $i/(2n)$ ensures that we choose the minimal $i$). Since $i_k^*$ is a minimum witness of $\breve{h}(k)$, the algorithm correctly computes $\breve{h}(k)$ for all $k \in [n+m]$. ◀

We remark that $\textsc{MinConv}(\breve{f}, \breve{g})$ could be computed using the SMAWK algorithm [1]. The reason we give a direct algorithm, is that we will need the witness path $P_0^-$ as computed by Algorithm 2.

**Computing $h = \textsf{MinConv}(f, g)$.** Recall that $f\colon [n] \mapsto \mathbf{Z}$ and $g\colon [m] \mapsto \mathbf{Z}$. As a final simplification, we argue that we can assume without loss of generality that $n = m$, and $n+1$ is a power of 2. To this end, let $N$ be the smallest power of 2 greater than $\max\{n, m\}$. We pad the functions to length $N$ by setting $f(n+j) := 2j \cdot W$ for $j \in [1\,..\,N-1-n]$ and $g(m+j) := 2j \cdot W$ for $j \in [1\,..\,N-1-m]$, where $W$ is an integer larger than $\max_{i \in [n]} f(i) + \max_{j \in [m]} g(j)$. Observe that the entries $h(0), \ldots, h(n+m)$ of the result $h = \textsc{MinConv}(f, g)$ are unchanged (due to the choice of sufficiently large $W$), so we can read off the original result from the result of the padded functions. Moreover, observe that the padding does not change the parameters $\Delta_f$ and $\Delta_g$.

Now we can describe the algorithm. After running Algorithm 2 we can assume that we have computed $\breve{h}$ and the witness path $P_0^- = \{(i_k^*, k - i_k^*) \mid k \in [n+m]\}$. We will make use of the following subroutines:

- $\textsc{Relevant}(i, j)$: returns $\breve{f}(i) + \breve{g}(j) \le \breve{h}(i+j) + 2\Delta$.
- $\textsc{BelowWitnessPath}(i, j)$: returns $i < i_{i+j}^*$
- $\textsc{AboveWitnessPath}(i, j)$: returns $i > i_{i+j}^*$

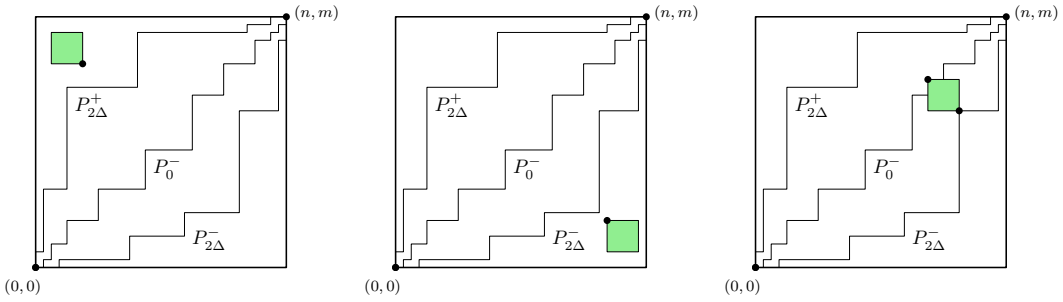
Now we can compute $h = \textsc{MinConv}(f, g)$ by calling $\textsc{RecMinConv}([0\,..\,n], [0\,..\,m])$. See Algorithm 3 for the pseudocode.

**Algorithm 3** Given intervals $I = [\, i_A \,..\, i_B \,], J = [\, j_A \,..\, j_B \,]$, the algorithm computes the contribution of $f[I]$ and $g[J]$ to $\textsc{MinConv}(f, g)$.

```
1:  procedure RecMinConv(I = [ i_A .. i_B ], J = [ j_A .. j_B ])
2:      if AboveWitnessPath(i_A, j_B) and NotRelevant(i_A, j_B) then        ▷ Case 1
3:          return h̃(k) = ∞ for all k ∈ [ i_A + j_A .. i_B + j_B ]
4:      if BelowWitnessPath(i_A, j_B) and NotRelevant(i_B, j_A) then        ▷ Case 2
5:          return h̃(k) = ∞ for all k ∈ [ i_A + j_A .. i_B + j_B ]
6:      if Relevant(i_A, j_B) and Relevant(i_B, j_A) then                   ▷ Case 3
7:          Compute C ← { (i, f(i)) | i ∈ I } + { (j, g(j)) | j ∈ J } using Corollary 19
8:          Infer h̃(k) ← min{ y | (k, y) ∈ C } for all k ∈ [ i_A + j_A .. i_B + j_B ]
9:          return h̃
10:     else                                                                ▷ Case 4
11:         Split I into two intervals I_1, I_2 of equal length, similarly split J into J_1, J_2
12:         Recursively compute g̃_{i,j} ← RecMinConv(I_i, J_j) for i, j ∈ { 1, 2 }
13:         return the pointwise minimum of the functions g̃_{i,j} for i, j ∈ { 1, 2 }
```

Algorithm 3 recursively computes the contribution of $f[\, i_A \,..\, i_B \,]$ and $g[\, j_A \,..\, j_B \,]$ to $h = \textsc{MinConv}(f, g)$. We next discuss its four cases; see Figure 2 for illustrations of Cases 1-3. If $(i_A, j_B)$ is above the witness path $P_0^-$ and is not $2\Delta$-relevant (Case 1), then as we argue below no point in $I \times J$ contributes to the output $h$, so in this case we return a dummy function (which is $+\infty$ everywhere). Case 2 is symmetric, where $(i_B, j_A)$ is above $P_0^-$ and not $2\Delta$-relevant, and we again return a dummy function. Case 3 applies when $(i_A, j_B)$ and $(i_B, j_A)$ are both $2\Delta$-relevant. In this case, we explicitly compute $\tilde{h} = \textsc{MinConv}(f[\, i_A \,..\, i_B \,], g[\, j_A \,..\, j_B \,])$ by computing the sumset $C = \{ (i, f(i)) \mid i \in I \} + \{ (j, g(j)) \mid j \in J \}$ and infering $\tilde{h}(k)$ as the minimum $y$ such that $(k, y) \in C$, which by definition of the sumset equals the minimum $f(i) + g(j)$ such that $i \in I, j \in J$ and $i + j = k$. Note that this step can be done for all $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$ in total time $O(|C|)$ by once scanning over all elements of $C$.

Finally, if none of the above cases apply, then we split both intervals $I$ and $J$ into equal halves and recurse on all 4 combinations of halves. We combine them by taking the pointwise minimum of all computed functions. This case is essentially brute force.




**(a)** Case 1. **(b)** Case 2. **(c)** Case 3.

**Figure 2** Visualization of Cases 1-3. The green box represents the current subproblem.

**Correctness.** We start by analyzing the correctness of the algorithm.

▶ **Lemma 21** (Correctness of Algorithm 3). *$\textsc{RecMinConv}([\, 0 \,..\, n \,], [\, 0 \,..\, m \,])$ (Algorithm 3) correctly computes $h = \textsc{MinConv}(f, g)$.*

**Proof.** Let $k \in [\, n+m \,]$ and consider a point $(i^*, j^*)$ in diagonal $k$ such that $f(i^*) + g(j^*) = h(k)$, i.e., a witness for $h(k)$. We argue that some recursive call computes $f(i^*) + g(j^*)$. This is clear in Case 4, as $(i^*, j^*)$ is covered by one recursive subproblem. It is also clear in Case 3, since then $f(i^*) + g(j^*)$ is explicitly computed.

To finish correctness, we argue that $(i^*, j^*)$ can never be in a subproblem to which Case 1 or 2 applies. Recall that Case 1 applies to a subproblem $I = [\, i_A \,..\, i_B \,], J = [\, j_A \,..\, j_B \,]$ if $(i_A, j_B)$ is above $P_0^-$ and $(i_A, j_B)$ is not $2\Delta$-relevant. Since $(i_A, j_B)$ is not $2\Delta$-relevant, by Lemma 17 $(i_A, j_B)$ must be above $P_{2\Delta}^+$ or below $P_{2\Delta}^-$. Since $(i_A, j_B)$ is above $P_0^-$, it can only be above $P_{2\Delta}^+$. Since $(i_A, j_B)$ is the lower right corner of $I \times J$, it follows that all points in $I \times J$ are above $P_{2\Delta}^+$. Thus, by Lemma 17 all points in $I \times J$ are not $2\Delta$-relevant. If we assume for the sake of contradiction that $(i^*, j^*) \in I \times J$, then Lemma 15 implies $f(i^*) + g(j^*) > h(k)$, contradicting the choice of $(i^*, j^*)$ as a witness for $h(k)$. Hence, $(i^*, j^*)$ can never be in a Case 1 subproblem. Case 2 is symmetric. This finishes the correctness proof. ◀

**Running Time.** Next, we analyze the running time. The key insight is that in relevant regions both functions are essentially linear, with the same slope (see Lemma 22). This implies that the sumset computed in Case 3 is small (see Lemma 23), so it can be computed efficiently using Corollary 19. In the following two lemmas, let $I = [\, i_A \,..\, i_B \,] \subseteq [\, n \,]$ and $J = [\, j_A \,..\, j_B \,] \subseteq [\, m \,]$ be intervals of the same length $|I| = |J|$.

▶ **Lemma 22** (Near Linearity, proof deferred to the full version). *If $I \times J \subseteq R_{2\Delta}$ then there are $a, b, c \in \mathbf{R}$ such that $|f(i) - (a \cdot i + b)| \leq 2\Delta$ for all $i \in I$ and $|g(j) - (a \cdot j + c)| \leq 2\Delta$ for all $j \in J$.*

▶ **Lemma 23** (Relevant Regions have Small Sumsets). *If $I \times J \subseteq R_{2\Delta}$ then the sumset $\{\, (i, f(i)) \mid i \in I \,\} + \{\, (j, f(j)) \mid j \in J \,\}$ has size $O(\Delta \cdot (|I| + |J|))$.*

**Proof.** By Lemma 22, for any $(i, j) \in I \times J$ with $i + j = k$ we have

$$f(i) + g(j) = (a \cdot i + b) + (a \cdot j + c) \pm O(\Delta) = a \cdot k + b + c \pm O(\Delta).$$

Thus, for each of the $|I| + |J| - 1$ $x$-coordinates (i.e., choices of $i + j$), there are $O(\Delta)$ different $y$-coordinates (i.e., values $f(i) + g(j)$) in the sumset. ◀

▶ **Lemma 24** (Running Time of Algorithm 3). *RecMinConv$([\, 0 \,..\, n \,], [\, 0 \,..\, m \,])$ (Algorithm 3) runs in time $\widetilde{O}(n\Delta)$.*

**Proof.** We first analyze the running time of one recursive subproblem, ignoring the cost of recursive calls. Note that in Cases 1 and 2 it suffices to return a dummy value, i.e., we do not need to iterate over $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$ to explitly return $\tilde{h}(k) = \infty$. Thus, Cases 1 and 2 run in time $O(1)$. We charge this time to the parent of the current subproblem, which is a Case 4-subproblem.

Consider Case 4. Ignoring the cost of the recursive subproblems, Case 4 runs in time $O(1)$, which also covers the charging from children which fall in Cases 1 and 2.

Consider Case 3, and let $s := i_B - i_A + 1 = j_B - j_A + 1$ be the current side length. By Lemma 23, the sumset computed in Algorithm 3 has size $O(\Delta s)$. Thus, it can be computed in time $\widetilde{O}(\Delta s)$ using Corollary 19, and the function $\tilde{h}$ can be inferred from it in time $O(\Delta s)$.

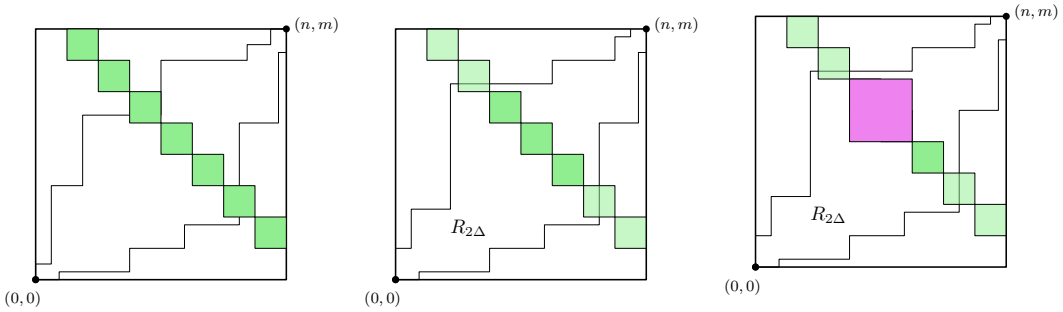
Now we bound the total running time across subproblems. Fix a side length $s$ and consider all possible subproblems of side length $s$, i.e., all boxes

$$B_{x,y}^s := [\, x \cdot s \,..\, x \cdot s + s - 1 \,] \times [\, y \cdot s \,..\, y \cdot s + s - 1 \,], \text{ where } x, y \in [\, n/s \,].$$

Consider a diagonal $D_{s,d} := \{ B^s_{x,x+d} \mid x \in [n/s] \}$ of these boxes, see Figure 3a. Note that a box in $D_{s,d}$ that lies fully above $P^+_\Delta$ corresponds to a Case 1-subproblem. A box in $D_{s,d}$ that lies fully below corresponds to a Case 2-subproblem. A box that is below or on $P^+_{2\Delta}$ and above or on $P^-_{2\Delta}$ corresponds to a Case 3-subproblem. The remaining boxes intersect $P^+_{2\Delta}$ or $P^-_{2\Delta}$ and correspond to Case 4.

Note that by monotonicity of $P^+_{2\Delta}, P^-_{2\Delta}$, at most two boxes in $D_{s,d}$ are intersected by $P^+_{2\Delta}$ or $P^-_{2\Delta}$ and thus at most two boxes in $D_{s,d}$ can appear as Case 4-subproblems. Thus, Case 4 incurs time $O(1)$ per diagonal. We argue that among the boxes in $D_{s,d}$, at most two can appear as Case 3-subproblems. Indeed, if these would be at least three such boxes, then the parent of the middle box would also be between $P^+_{2\Delta}$ and $P^-_{2\Delta}$, and thus the parent would already be a Case 3-subproblem, see Figures 3b and 3c. Thus, the middle box would not get split, and it would not become a recursive subproblem. Hence per diagonal $D_{s,d}$, Case 3 incurs time $\widetilde{O}(\Delta s)$ for each of at most two boxes.

It remains to sum up over all side lengths $1 \leq s \leq n$ where $s = 2^\ell$ is a power of 2 (recall that at each recursive level we split the side length in two equal parts), and over all $O(n/s)$ diagonals $d$, to obtain total time $\sum_{\ell=1}^{\log n} O(n/2^\ell) \cdot \widetilde{O}(\Delta 2^\ell) = \widetilde{O}(\Delta n)$. Note that the sum over $\ell$ only adds another log-factor, which is hidden by the $\widetilde{O}$-notation. ◀




**(a)** A diagonal of boxes $D_{s,d}$.

**(b)** Three boxes inside $R_{2\Delta}$.

**(c)** The parent box is already contained in $R_{2\Delta}$.

**Figure 3** Visualizations for the proof of Lemma 24.

## References


1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In *SOSA*, pages 57–67. SIAM, 2021. doi:10.1137/1.9781611976496.6.

3 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster Knapsack and graph algorithms. In *ICALP*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.19.

4 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *STOC*, pages 1269–1282. ACM, 2018. doi:10.1145/3188745.3188876.

5 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957. doi:10.2307/j.ctv1nxcw0f.

6 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/s00453-012-9734-3.

7 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *SODA*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

8 Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In *ICALP*, volume 229 of *LIPIcs*, pages 31:1–31:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.31.

9 Karl Bringmann, Nick Fischer, and Vasileios Nakos. Sparse nonnegative convolution is equivalent to dense nonnegative convolution. In *STOC*, pages 1711–1724. ACM, 2021. doi:10.1145/3406325.3451090.

10 Karl Bringmann, Nick Fischer, and Vasileios Nakos. Deterministic and las vegas algorithms for sparse nonnegative convolution. In *SODA*, pages 3069–3090. SIAM, 2022. doi:10.1137/1.9781611977073.119.

11 Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In *SODA*, pages 1797–1815. SIAM, 2021. doi:10.1137/1.9781611976465.108.

12 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *SODA*, pages 1777–1796. SIAM, 2021. doi:10.1137/1.9781611976465.107.

13 Michael R. Bussieck, Hannes Hassler, Gerhard J. Woeginger, and Uwe T. Zimmermann. Fast algorithms for the maximum convolution problem. *Oper. Res. Lett.*, 15(3):133–141, 1994. doi:10.1016/0167-6377(94)90048-5.

14 Timothy M. Chan. Approximation schemes for 0-1 knapsack. In *SOSA*, volume 61 of *OASIcs*, pages 5:1–5:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.5.

15 Timothy M. Chan and Qizheng He. More on change-making and related problems. *J. Comput. Syst. Sci.*, 124:159–169, 2022. doi:10.1016/j.jcss.2021.09.005.

16 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *STOC*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

17 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

18 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

19 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.

20 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

21 Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In *SODA*, pages 2961–2979. SIAM, 2023. doi:10.1137/1.9781611977554.ch113.

22 Mingyang Deng, Xiao Mao, and Ziqian Zhong. On problems related to unbounded subsetsum: A unified combinatorial approach. In *SODA*, pages 2980–2990. SIAM, 2023. doi:10.1137/1.9781611977554.ch114.

23 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. In *SODA*, pages 808–816. SIAM, 2018. doi:10.1137/1.9781611975031.52.

24 Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. Essentially optimal sparse polynomial multiplication. In *ISSAC*, pages 202–209. ACM, 2020. doi:10.1145/3373207.3404026.

25 Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

**26** Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *ITCS*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.43.

**27** Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *SOSA*, volume 69 of *OASIcs*, pages 17:1–17:6. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.17.

**28** Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004. doi:10.1023/B:JOCO.0000021934.29833.6b.

**29** Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi:10.1007/978-3-540-24777-7.

**30** Kim-Manuel Klein. On the fine-grained complexity of the unbounded subsetsum and the frobenius problem. In *SODA*, pages 3567–3582. SIAM, 2022. doi:10.1137/1.9781611977073.141.

**31** Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.

**32** Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *ICALP*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

**33** Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *IEEE Trans. Inf. Theory*, 66(11):7231–7236, 2020. doi:10.1109/TIT.2020.2989385.

**34** David Pisinger. Linear time algorithms for Knapsack problems with bounded weights. *J. Algorithms*, 33(1):1–14, 1999. doi:10.1006/jagm.1999.1034.

**35** Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and Subset Sum with small items. In *ICALP*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.106.

**36** R. Ryan Williams. Faster All-Pairs Shortest Paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

# Funnelselect: Cache-Oblivious Multiple Selection

**Gerth Stølting Brodal**
Aarhus University, Denmark

**Sebastian Wild**
University of Liverpool, UK

## Abstract

We present the algorithm *funnelselect*, the first optimal randomized cache-oblivious algorithm for the multiple-selection problem. The algorithm takes as input an unsorted array of $N$ elements and $q$ query ranks $r_1 < \cdots < r_q$, and returns in sorted order the $q$ input elements of rank $r_1, \ldots, r_q$, respectively. The algorithm uses expected and with high probability $O\left(\sum_{i=1}^{q+1} \frac{\Delta_i}{B} \cdot \log_{M/B} \frac{N}{\Delta_i} + \frac{N}{B}\right)$ I/Os, where $B$ is the external memory block size, $M \geq B^{1+\varepsilon}$ is the internal memory size, for some constant $\varepsilon > 0$, and $\Delta_i = r_i - r_{i-1}$ (assuming $r_0 = 0$ and $r_{q+1} = N + 1$). This is the best possible I/O bound in the cache-oblivious and external memory models. The result is achieved by reversing the computation of the cache-oblivious sorting algorithm funnelsort by Frigo, Leiserson, Prokop and Ramachandran [FOCS 1999], using randomly selected pivots for distributing elements, and pruning computations that with high probability are not expected to contain any query ranks.

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

**Keywords and phrases** Multiple selection, cache-oblivious algorithm, randomized algorithm, entropy bounds

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.25

**Funding** *Gerth Stølting Brodal*: Independent Research Fund Denmark, grant 9131-00113B.

## 1 Introduction

We present the first optimal randomized cache-oblivious algorithm for the multiple-selection problem. Our result combines ideas from the cache-oblivious sorting algorithm funnelsort with existing multiple-selection algorithms. Many existing time- and comparison-optimal multiple-selection algorithms are already cache oblivious, but they are not optimal with respect to the number of I/Os performed when analyzed in the cache-oblivious model.

Let us start with a brief history of the multiple-selection problem. In 1961, Hoare presented the classic randomized sorting algorithm *quicksort*, published as Algorithm 64 in the Algorithms column of the Communications of the ACM [15]. Quicksort makes essential use of the randomized algorithm *partition* (Algorithm 63 [14]), that picks a random element, denoted a *pivot*, and partitions the elements into those smaller and larger than the pivot. By recursing on each subproblem, quicksort sorts an input of size $N$ in expected $O(N \lg N)$ time and comparisons[1]. Hoare observed that if we are only interested in finding the $r$th smallest element in the input, denoted the element of *rank* $r$, we do not need to sort the input completely. By pruning recursive calls in quicksort not relevant for finding the $r$th smallest element, the resulting algorithm *find* (Algorithm 65 [16]) achieves expected $O(N)$ time. Chambers [7] generalized this idea to finding $q$ elements of $q$ given ranks $1 \leq r_1 < r_2 < \cdots < r_q \leq N$, in the following denoted the *multiple-selection* problem, by just skipping all recursive problems not containing any query rank. The expected running time is $O(N \lg q)$, but Prodinger [19] proved a tighter expected bound of $O(\mathcal{B} + N)$, where $\mathcal{B} = \sum_{i=1}^{q+1} \Delta_i \lg \frac{N}{\Delta_i}$ with $\Delta_i = r_i - r_{i-1}$, for $1 \leq i \leq q+1$, assuming $r_0 = 0$ and $r_{q+1} = N+1$.

[1] lg denotes the binary logarithm.





We call $\mathcal{B}$ the *entropy* of the multiple-selection query [3]. Dobkin and Munro [8] achieved matching asymptotic bounds for the worst-base time in the comparison model by using a deterministic linear-time (single) selection algorithm [4, 20] for the partitioning steps.

## 1.1 Model of Computation

In this paper we study the multiple-selection problem in a hierarchical-memory model, where we have an infinite external memory and an internal memory of capacity $M$ elements, and where data is transferred between the internal and external memory in blocks of $B$ consecutive elements. A block transfer is called an *I/O* (input/output operation). The I/O cost of an algorithm is the number of I/Os it performs. Aggarwal and Vitter [1] introduced this as the *external-memory model* and proved that sorting in this model requires $\Theta\big(\frac{N}{B}\log_{M/B}\frac{N}{B}\big)$ I/Os. The upper bound is, e.g., achieved by $M/B$-way mergesort and distributionsort algorithms, where the algorithms exploit knowledge of the parameters $M$ and $B$.

Frigo *et al.* [12, 13] introduced the *cache-oblivious model*, that essentially is the same as the external-memory model, except that algorithms *do not know* $M$ and $B$, and I/Os are assumed to be performed automatically by an optimal paging algorithm. As a consequence, cache-oblivious algorithms also adapt to multi-level memory hierarchies (under certain conditions [13]). The same paper introduced the cache-oblivious sorting algorithm *funnelsort* achieving the optimal external-memory I/O bound, assuming a *"tall-cache"*, $M = \Omega(B^2)$. Brodal and Fagerberg [6] observed that under the weaker tall-cache assumption $M \geq B^{1+\varepsilon}$, for a constant $\varepsilon > 0$, the optimal I/O bound increases by a factor $\Theta(1/\varepsilon)$.

Multiple selection was studied in external-memory by Hu *et al.* [17] and Barbay *et al.* [3]. The algorithms have an I/O cost of $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$, where $\mathcal{B}_{\mathrm{I/O}} = \frac{\mathcal{B}}{B\lg(M/B)}$. A matching lower bound was sketched in [3] as a reduction from sorting, assuming the multiple-selection algorithm partitions the input elements into the gaps between the queried elements (most algorithms actually solve this problem, that Chambers denoted *partial sorting*). Hu *et al.* [17] considered the case where the queried elements can be returned in arbitrary order without partial sorting, and showed that without a tall-cache assumption, this problem can actually be solved asymptotically faster for a small number of queries $q$.

## 1.2 Results

Our first result is a lower bound for the external-memory multiple-selection problem (and not only for the partial-sorting problem as in the lower bound of Barbay *et al.* [3]).

▶ **Theorem 1** (Lower bound). *External-memory multiple selection in expectation requires* $\Omega(\mathcal{B}_{\mathrm{I/O}}) - O\big(\frac{N}{B}\log_{M/B} B\big)$ *I/Os.*

Note that an external-memory lower bound is also a cache-oblivious lower bound (for any online paging strategy), and that under a tall-cache assumption $M \geq B^{1+\varepsilon}$, for a constant $\varepsilon > 0$, the last term $O\big(\frac{N}{B}\log_{M/B} B\big) = O\big(\frac{1}{\varepsilon}\cdot\frac{N}{B}\big)$. The result is obtained by combining the comparison lower bound for the multiple-selection problem by Dobkin and Munro [8] with the general reduction technique of Arge *et al.* [2], that can derive an I/O-decision-tree lower bound from a comparison-decision-tree lower bound.

Our second result is the cache-oblivious algorithm funnelselect.

▶ **Theorem 2** (Funnelselect upper bound). *There exists a randomized cache-oblivious algorithm solving the multiple-selection problem using* $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ *I/Os in expectation and with high probability.*

**Table 1** Algorithms for selection and multiple selection. CO = cache-oblivious, $\mathbb{E}$ = expected, wc = worst-case bounds. Note that Barbay *et al.* assume a tall cache, whereas Hu *et al.* do not.

| Reference | | Comparisons | I/Os | Comments |
|---|---|---|---|---|
| ***Single selection*** | | | | |
| Hoare [16] | $\mathbb{E}$ | $2\ln 2\mathcal{B} + 2N + o(N)$ | $O(N/B)$ | CO, randomized |
| Floyd & Rivest [11] | $\mathbb{E}$ | $N + \min\{r, N-r\} + o(N)$ | $O(N/B)$ | CO, randomized |
| Blum *et al.* [4] | wc | $5.4305N$ | $O(N/B)$ | CO, deterministic |
| Schönhage *et al.* [20] | wc | $3N + o(N)$ | ? | deterministic, median |
| Dor & Zwick [9] | wc | $2.95 + o(N)N$ | ? | deterministic, median |
| ***Multiple selection*** | | | | |
| Chambers [7, 19] | $\mathbb{E}$ | $2\ln 2\mathcal{B} + O(N)$ | $O((\mathcal{B}+N)/B)$ | CO, randomized |
| Dobkin & Munro [8] | wc | $3\mathcal{B} + O(N)$ | $O((\mathcal{B}+N)/B)$ | CO, deterministic |
| Kaligosi *et al.* [18] | wc | $\mathcal{B} + o(\mathcal{B}) + O(N)$ | $O((\mathcal{B}+N)/B)$ | CO, deterministic |
| Hu *et al.* [17] | wc | $O(N\lg(q))$ | $O(N/B\log_{M/B}(q/B))$ | deterministic |
| | wc | $O(\mathcal{B}+N)$ | $O(\mathcal{B}_{\mathrm{I/O}} + N/B)$ | (from closer analysis) |
| Barbay *et al.* [3] | wc | $\mathcal{B} + o(\mathcal{B}) + O(N)$ | $O(\mathcal{B}_{\mathrm{I/O}} + N/B)$ | online, determ., $M \geq B^{1+\varepsilon}$ |
| *New* (Theorem 2) | $\mathbb{E}$ | $O(\mathcal{B}+N)$ | $O(\mathcal{B}_{\mathrm{I/O}} + N/B)$ | CO, randomized, $M \geq B^{1+\varepsilon}$ |

At the high level, the result is obtained by the standard approach of recursively partitioning by pivots and pruning computations not containing any query ranks. To achieve good I/O performance in the cache-oblivious model we pipeline the partitioning by essentially reversing the computations done by funnelsort, and replace each merging node by a partitioning node. Since we do not know the ranks of the pivots during the partitioning, we pick the pivots carefully from a random sample such that a concentration bound guarantees approximate ranks of the pivots, so we can truncate computations that with high probability do not contain any query ranks. Table 1 summarizes known and the new results.

## 1.3 Preliminaries and Notation

Throughout the paper we assume that the input to a multiple-selection algorithm are two arrays $S$ and $R$, where $S$ is an unsorted array of $N$ elements from a totally ordered universe, and $R$ is a sorted array $r_1, \ldots, r_q$ of $q$ distinct query ranks, where $1 \leq r_1 < \cdots < r_q \leq N$. Our task is to report an array of the $q$ order statistics $S_{(r_1)}, \ldots, S_{(r_q)}$, where $S_{(r)}$ is the $r$th smallest element in $S$, i.e., the element at index $r$ in an array storing $S$ after sorting it. If $x$ is an element and $S$ a set, we let $x < S$ denote that $x < y$ for all $y$ in $S$. Unless stated otherwise, we assume that all elements in $S$ are distinct.

## 1.4 Outline of Paper

In Section 2 we prove the I/O lower bound for multiple selection stated in Theorem 1. In Section 3 we present internal-memory and external-memory algorithms as a warm-up for the cache-oblivious algorithm in Section 4 achieving Theorem 2. In Section 5 we analyze the algorithm. In Section 6, we discuss how to extend the algorithm to partially sort the input, and in Section 7, we discuss how to deal with equal elements. Section 8 concludes with open problems.

# 2 Lower Bound

In this section we prove Theorem 1. Dobkin and Munro [8, Theorem 1] observed that the comparisons done by a comparison-based multiple-selection algorithm must classify the remaining elements into "*gaps*" between the selected elements, and by sorting each of these

gaps with $\Delta_i - 1$ elements using $\Delta_i \lg \Delta_i - O(\Delta_i)$ additional comparisons, one can sort the input. Together with the $N \lg N - O(N)$ lower bound on comparison based sorting, we have

$$\#\text{comparisons for multiple selection} + \sum_{i=1}^{q+1}(\Delta_i \lg \Delta_i - O(\Delta_i)) \;\geq\; N \lg N - O(N)\,,$$

implying a lower bound of $\mathcal{B} - O(N)$ on the number of comparisons for multiple selection, where $\mathcal{B} = \sum_{i=1}^{q+1} \Delta_i \lg\left(\frac{N}{\Delta_i}\right)$. This holds for worst, average, and expected case.

To prove I/O lower bounds on external-memory algorithms, Arge *et al.* [2] presented a general reduction that converts a comparison lower bound into an I/O lower bound, by converting an I/O-decision tree $T$ to a standard comparison decision tree $T_c$. An I/O-decision tree consists of unary I/O-nodes moving $B$ elements between internal and external memory, and comparison nodes between two elements in internal memory. Their lower bound reduction [2, Corollary 5] relates for any input $x$, the number of I/Os in $T$, $\#\text{I/Os}_T(x)$, to the number of comparisons in $T_c$, $\#\text{comparisons}_{T_c}(x)$, as

$$\#\text{comparisons}_{T_c}(x) \;\leq\; N \lg B + \#\text{I/Os}_T(x) \cdot B\left(3 + \lg \frac{M-B}{B}\right). \tag{1}$$

Since the reduction relates comparisons and I/Os for each input instance, the reduction can be used to show worst-case, average-case, and expected-case lower bounds.

Plugging the $\mathcal{B} - O(N)$ comparison lower bound into eq. (1) we get

$$\sum_{i=1}^{q+1} \Delta_i \lg \frac{N}{\Delta_i} - O(N) \;\leq\; N \lg B + \#\text{I/Os} \cdot B\left(3 + \lg \frac{M-B}{B}\right),$$

implying the following I/O lower bound for multiple selection:

$$\#\text{I/Os} \;\geq\; \frac{1}{1 + \frac{3}{\lg(M/B)}} \cdot \mathcal{B}_{\text{I/O}} - O\left(\frac{N}{B} \log_{M/B} B\right) \;=\; \Omega(\mathcal{B}_{\text{I/O}}) - O\left(\frac{N}{B} \log_{M/B} B\right),$$

for $M \geq 2B$ and $\mathcal{B}_{\text{I/O}} = \sum_{i=1}^{q+1} \frac{\Delta_i}{B} \log_{M/B} \frac{N}{\Delta_i} = \frac{\mathcal{B}}{B \lg(M/B)}$. This concludes the proof of Theorem 1.

Aggarwal and Vitter [1, Theorem 3.1] proved that comparison-based external-memory sorting requires $\Omega\left(\frac{N}{B} \cdot \log_{M/B} \frac{N}{B}\right)$ I/Os. This lower bound also applies to sorting in the cache-oblivious model. Brodal and Fagerberg [6, Corollary 2] showed that for a cache-oblivious sorting algorithm to be asymptotically optimal for all choices of $M$ and $B$, a "tall-cache" assumption $M \geq B^{1+\varepsilon}$ is *necessary*. Since we can sort $N$ elements using a multiple-selection algorithm by querying all ranks $1, \ldots, N$, a tall-cache assumption is also necessary for matching bounds for multiple selection in the cache-oblivious model.

## 3 Internal-Memory and External-Memory Multiple Selection

In this section we consider simple internal-memory and external-memory (cache-conscious) algorithms for multiple selection as a warm-up for our cache-oblivous algorithm in Section 4, which borrows ideas from both algorithms.

### 3.1 Internal Memory

A simple recursive internal-memory algorithm is MultiSelect (Algorithm 1). This is essentially Chamber's algorithm from 1971 [7], except for the choice of pivot. If there are no query ranks in $R$, nothing needs to be reported. Otherwise, pick a pivot $P$ from $S$, partition

**Algorithm 1** Internal-memory multiple selection.

```
1: procedure MultiSelect(S[1..N], R[1..q])
2:     if R ≠ ∅ then
3:         P ← median of S (pivot)
4:         Partition S into S_1 < P < S_2
5:         r̄ ← |S_1| + 1 (the rank of P in S)
6:         Partition R into R_1 < r̄ < R_2
7:         MultiSelect(S_1, R_1)
8:         if r̄ ∈ R then
9:             Report P
10:        MultiSelect(S_2, {r − r̄ | r ∈ R_2})
```

**Algorithm 2** External-memory multiple selection (multi-way generalization of MultiSelect).

```
1: procedure MultiSelect_I/O(S[1..N], R[1..q])
2:     if R ≠ ∅ then
3:         Find k̄ − 1 ≤ k − 1 pivots P_1 < ··· < P_{k̄−1} in S
4:         Partition S into S_1, ..., S_k̄ s.t. P_{i−1} < S_i < P_i (P_0 = −∞, P_k̄ = +∞)
5:         r̄_i ← i + |S_1| + ··· + |S_i| (the rank of P_i in S)
6:         Partition R into R_1, ..., R_k̄ s.t. r̄_{i−1} < R_i < r̄_i (r̄_0 = 0, r̄_k̄ = N + 1)
7:         for i = 1, ..., k̄ do
8:             MultiSelect_I/O(S_i, {r − r̄_{i−1} | r ∈ R_i})
9:             if r̄_i ∈ R then
10:                Report P_i
```

$S \setminus \{P\}$ into $S_1$ and $S_2$, such that $S_1 < P < S_2$, compute the rank $\bar{r}$ of the pivot $P$ in $S$, partition $R \setminus \{\bar{r}\}$ into $R_1$ and $R_2$, such that $R_1 < \bar{r} < R_2$, and recurse on the subproblems $(S_1, R_1)$ and $(S_2, R_2)$. The pivot $P$ is output before the second recursion if $\bar{r}$ is a query rank in $R$ (so elements are reported in increasing rank order). This intuitively corresponds to a distributionsort/quicksort, where we truncate recursive calls not containing any query ranks in $R$.

In Algorithm 1, $P$ is the exact median of $S$, but we could also have used an approximate median, or a randomly sampled pivot. Chamber's original algorithm uses a random element from $S$. Finding the pivot can be done using the deterministic linear-time median finding algorithms by Blum *et al.* [4] or the randomized algorithms by Hoare [16] or Floyd and Rivest [11]. Prodinger [19] proved that selecting a random pivot leads to expected overall $O(\mathcal{B} + N)$ time. Kaligosi *et al.* [18, Section 2] proved that Algorithm 1 achieves $O(\mathcal{B} + N)$ worst-case time, if a linear time median selection algorithm is used.

Algorithm MultiSelect is cache oblivious, since it is designed independently of the memory parameter $B$ and $M$. All the above median algorithms are based on repeatedly scanning arrays and (analyzed in the cache-oblivious model) require $O(N/B)$ I/Os worst-case and expected, respectively. Since the additional work of MultiSelect can be implemented by repeatedly scanning arrays allocated on a stack, the I/O cost of the algorithm equals the internal computation time divided by the external-memory block size, i.e., $O(\mathcal{B}/B)$ I/Os. Our cache-oblivious algorithm from Section 4 improves upon this I/O cost by a factor $\Theta\big(\lg \frac{M}{B}\big)$.

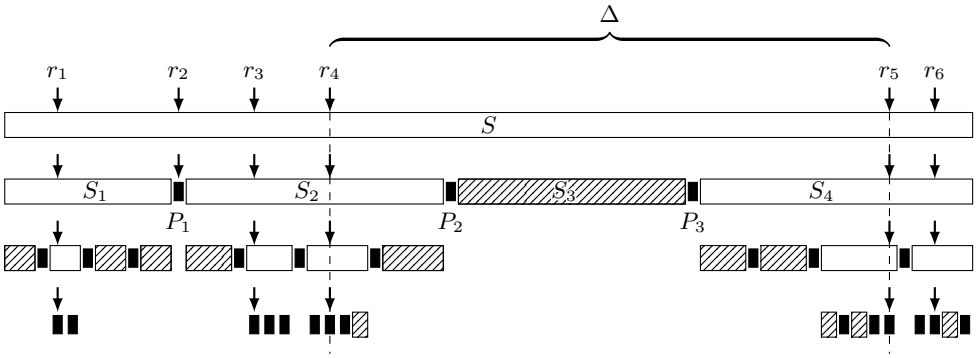



**Figure 1** Recursion for MultiSelect$_{I/O}$ with six query ranks and $k = 4$. Black squares are pivots, arrows show rank queries, and shaded areas are skipped subproblems with no query ranks.

## 3.2 External Memory

A generalization of MultiSelect better suited for external memory is to replace the binary partitioning by a multi-way partitioning. For a parameter $k \geq 2$, we assume the set $S$ is partitioned into $\bar{k} \leq k$ subsets around $\bar{k} - 1$ pivots, where each set has size $O(|S|/k)$. The resulting algorithm is shown as MultiSelect$_{I/O}$ in Algorithm 2. Figure 1 shows a recursion for MultiSelect$_{I/O}$ on an example with six query ranks. If all sets $S_i$ defined by the pivots have size at most $\alpha|S|$, where $1/k \leq \alpha < 1$, we denote the partitioning a $(k, \alpha)$-*partitioning*. The algorithm MultiSelect (with exact medians) is the special case of MultiSelect$_{I/O}$, where we use a $\left(2, \frac{1}{2}\right)$-partitioning.

There are several $(k, O(1/k))$-partitioning schemes described in the literature, e.g., a $(k, 1.5/k)$-partitioning method with $k = \sqrt{M/B}$ by Aggarwal and Vitter [1]. Here we describe a simpler $(k, 2/k)$-partitioning that incrementally inserts the $N$ elements of $S$ into buckets defined by a monotonically growing set of pivots, that also works for $k = \Theta(M/B)$. Initially there is one empty bucket and no pivot. Whenever a bucket reaches size $> 2N/k$ (i.e., the size is $1 + \lfloor 2N/k \rfloor$), the median of the bucket is selected as a new pivot, and the bucket is split around the pivot into two buckets with the elements smaller than and larger than the new pivot, respectively. Each new bucket has size at least $\lfloor N/k \rfloor$. Therefore, the total number of buckets created is at most $k$ and each bucket contains at most $2N/k$ elements.

Crucial for the I/O effectiveness of this partitioning is that one memory block from each bucket is in memory while scanning $S$ and distributing elements to buckets, i.e., $k \leq c\frac{M}{B}$ for a suitable constant $0 < c < 1$. Since each bucket can be split using $O(N/(kB))$ I/Os using the deterministic selection algorithm from [4], the total cost for creating a $(k, 2/k)$-partitioning of $S$ is $O(N/B)$ I/Os, provided $k \leq c\frac{M}{B}$. A binary search to find the bucket for an element requires $\leq \lceil \lg(k-1) \rceil$ comparisons with pivots, i.e., in total $O(N \lg k)$ comparisons for distributing to buckets. Since each of the at most $k - 1$ bucket splits requires $O(N/k)$ comparisons [4], creating a $(k, 2k)$-partitioning requires $O(N \lg k)$ comparisons.

## 3.3 Analysis

We now analyze the comparison and I/O cost of MultiSelect$_{I/O}$. Assume creating a $(k, \alpha)$-partitioning has an (abstract) cost of $C \cdot N$, where $C = C(k, \alpha, M, B)$ does not depend on $N$. For example, when counting comparisons we have $C = \Theta(\lg k)$. The total cost of MultiSelect$_{I/O}$ is the sum of the costs of all partitioning steps, i.e., $C$ times the sum of the sizes of all the subsets partitioned by the algorithm (the white rectangles in Figure 1).

Consider any fixed gap $\Delta$ between two query ranks. We assume that the right query rank (but not the left) is part of the gap, so that all elements of $S$ belong to exactly one gap. At each level of the recursion, the gap $\Delta$ intersects at most two subproblems that need to be partitioned, namely the subproblems containing the query ranks at the boundary of $\Delta$ (illustrated by the dashed lines in Figure 1). Since subproblems at depth $d$ of the recursion have size at most $\alpha^d N$, the total cost we need to charge within gap $\Delta$ is at most $C$ times

$$\sum_{d=0}^{\infty} \min(\Delta, 2\alpha^d N) \;\leq\; \Delta \left\lfloor \log_{1/\alpha} \frac{2N}{\Delta} \right\rfloor + \Delta \sum_{i=0}^{\infty} \alpha^i \;\leq\; \Delta \log_{1/\alpha} \frac{2N}{\Delta} + \frac{\Delta}{1-\alpha} \,.$$

Here we use that $2\alpha^d N$ is geometrically decreasing and $\Delta = 2\alpha^d N$ implies $d = \log_{1/\alpha} \frac{2N}{\Delta}$, i.e., in the sum, $\Delta$ is the term for the first $\lfloor \log_{1/\alpha} \frac{2N}{\Delta} \rfloor$ levels, whereas the remaining terms are geometrically decreasing, starting with at most $\Delta$. Summing over all gaps we obtain total cost at most

$$C \cdot \left( \sum_{i=1}^{q+1} \Delta_i \log_{1/\alpha} \frac{2N}{\Delta_i} + \frac{N}{1-\alpha} \right) . \tag{2}$$

Recall that MultiSelect is the special case of MultiSelect$_{\mathrm{I/O}}$ with $k = 2$ and $\alpha = 1/2$. For comparisons, we have $C = O(1)$ per processed element in partitioning. By eq. (2), the total number of comparisons in MultiSelect is $O(\mathcal{B} + N)$. For MultiSelect$_{\mathrm{I/O}}$ we have $k = cM/B$ and $\alpha = 2/k$, and a cost of $C = O(1/B)$ I/Os per processed element, so by eq. (2), MultiSelect$_{\mathrm{I/O}}$ has a total cost of $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os. Alternatively, using the multiway partitioning method of Aggarwal and Vitter [1] with $k = \sqrt{M/B}$, $\alpha = 1.5/k$, and cost $C = O(1/B)$ for I/Os, we also get a total cost of $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os from eq. (2).

## 4 Cache-Oblivious Multiple Selection

In this section we present our cache-oblivious multiple-selection algorithm FunnelSelect (Algorithm 4). We first recall funnels for merging (Section 4.1) and then show that they can be used for partitioning (Section 4.2). FunnelSelect performs a single round of such a funnel-based partitioning, splitting the input into $k$ parts of expected size $\Theta(N/k)$ using $k-1$ pivots, where $k = \Theta(N^{1/d})$ and $d = \max\{1 + 2/\varepsilon, 3\}$ under the tall-cache assumption $M \geq B^{1+\varepsilon}$. We then deal with each of the $k$ parts with a non-empty set of rank queries by fully sorting it and returning the sought ranks.

However, to stay within the allowed I/O bound, we have to truncate partitioning, namely whenever neither side of the split is likely to contain a query rank (Section 4.4). To boost the probability of "guessing correctly" which buckets query ranks fall into, we also have to choose pivots judiciously (Section 4.3). Section 5 then proves Theorem 2.

### 4.1 Funnelsort

Since our cache-oblivious multiple-selection algorithm is heavily based on ideas from the optimal cache-oblivious sorting algorithm funnelsort by Frigo *et al.* [12, Section 4], we briefly recall funnelsort and in particular its $k$-merger construction here.[2] Funnelsort uses

[2] It should be noted that the cache-oblivious distributionsort algorithm in [12, Section 5] is a significantly different approach than the one taken by funnelsort and our algorithm, even though our algorithm highly resembles a classic internal-memory distributionsort algorithm.

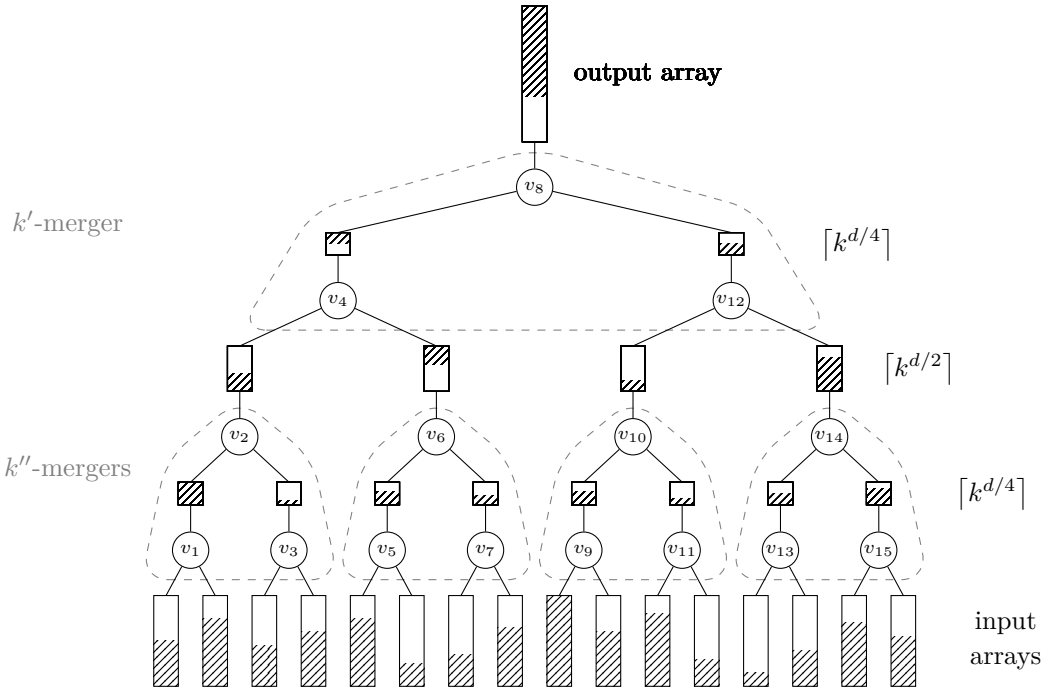



**Figure 2** A $k$-merger in funnelsort for $k = 16$ input arrays. Content in the buffers is shaded; elements are added to the bottom of buffers and consumed from the top of buffers. The figure shows the situation where $v_6$ is in the process of filling its output buffer, after being recursively called from $v_4$ during its merging, which in turn has been called by $v_8$ during its merging. Buffer sizes for the three internal levels are shown next to the buffers.

$O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os, assuming the tall-cache assumption $M \geq B^2$. Brodal and Fagerberg [5, Lemma 1] presented a lazy version of funnelsort, achieving the same I/O bound under the weaker tall-cache assumption $M \geq B^{1+\varepsilon}$, for any constant $\varepsilon > 0$. Funnelsort sorts an array of $N$ elements by an outer recursion that partitions the input into $k$ arrays each of size at most $\lceil N/k \rceil$, sorts these subarrays recursively, and then merges these arrays using a $k$-way construction named a *$k$-merger*. The parameter $k$ depends on the tall-cache assumption (via $\varepsilon$) and the input size $N$: $k = 2^{\lceil \lg(N)/d \rceil} = \Theta(N^{1/d})$ for $d = 1 + 2/\varepsilon$.

A $k$-merger (see Figure 2) consists of a perfectly balanced binary tree of height $\lg k$ of binary *merger-nodes*, where each tree edge contains a buffer that is a sorted array of elements. Each merger-node consumes elements from two child buffers and feeds into its parent output buffer. When invoking a merger-node $v$, the node $v$ fills its output buffer by merging the content of its input buffers until either the output buffer is full or one of the input buffers is exhausted. If the input buffer of a child $w$ is exhausted, we recursively invoke $w$ to fill its output buffer; then $v$ continues to fill its output buffer.

The I/O efficiency of funnelsort hinges entirely on a judicious choice of buffer sizes. The buffers connecting the middle levels of binary mergers (between level $\lceil \lg(k)/2 \rceil$ and one below) can hold $\lceil k^{d/2} \rceil$ elements each. The construction is recursively applied to a $k' = 2^{\lceil \lg(k)/2 \rceil} \approx \sqrt{k}$-merger forming the top $\lceil \lg(k)/2 \rceil$ levels and the $k'$ children each of which is a $k'' = 2^{\lfloor \lg(k)/2 \rfloor} \approx \sqrt{k}$-merger below the middle buffers; following a van-Emde-Boas layout of the binary tree and recursively allocating buffers consecutively in memory in that order.

## 4.2 Funnels for Partitioning

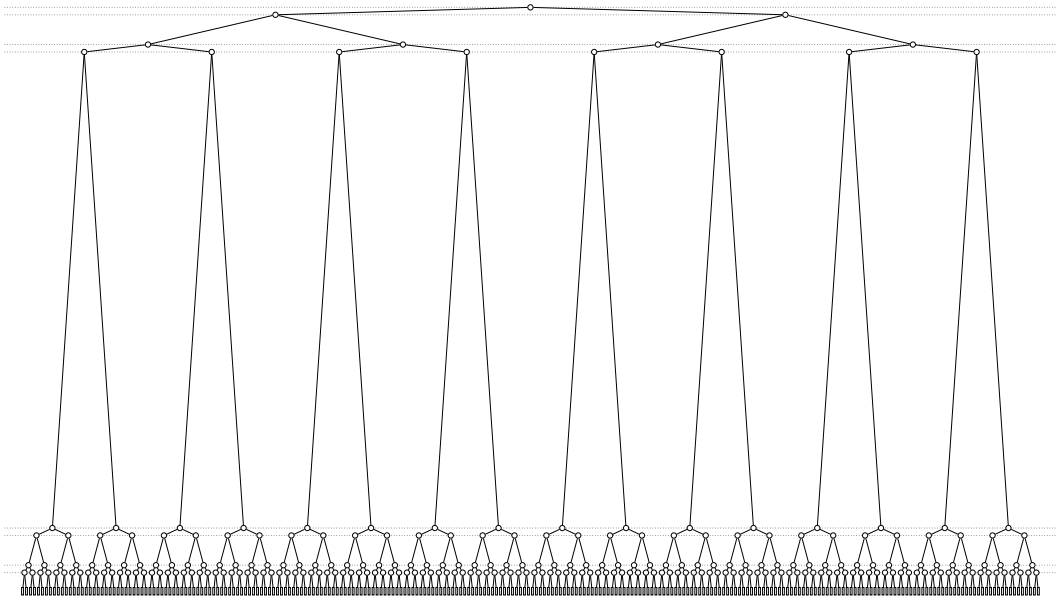


**Figure 3** A 256-partitioner; splitter nodes are shown as circles (as in Figure 2); buffers are only shown as edges, but with the vertical length of the edges to scale with the buffer sizes for $d = 2$. (Buffer sizes by level are 4, 16, 4, 256, 4, 16, 4).

A key innovation of this paper is the *k-partitioner*, which uses funnels *in reverse*: instead of merging $k$ runs, we push elements down the funnel while partitioning them around $k-1$ pivots into $k$ buckets. We use the same internal buffer sizes for a $k$-partitioner as in a $k$-merger; each buffer is organized as a queue and maintains an element count. The $k$ output buffers at the bottom of a $k$-partitioner are conceptually unbounded (never full). Note that in partitioning, we always know the number $N$ of elements and can allocate output buffers as linked lists of blocks of size $\Theta(N/k)$. Calling Partition (Algorithm 3) on a node $v$ partitions elements around $v$'s pivot and passes them to one of $v$'s children, recursively emptying these whenever they become full. FunnelPartition starts with all elements in the root's input buffer and then calls Partition on the root. After that, Flush recursively empties any remaining nonempty buffers.

Figure 2 can be read *mutatis mutandi* as a $k$-partitioner instead of a $k$-merger: Each node $v_i$ stores a pivot $P_i$ and Partition pushes elements towards the leaves. Buffers are consumed from the bottom and filled at the top of the hatched area. Figure 2 shows an overall Partition call at $v_8$, where first $v_4$ and then $v_6$ had run full; currently $v_6$ is moving elements from its parent buffer to its child buffers. Note that buffer sizes in Figure 2 are not drawn to scale; Figure 3 gives a more truthful representation.

The main property of $k$-partitioners is given in Lemma 3 below. It is similar to [6, Lemma 1], but we give a self-contained proof here.

▶ **Lemma 3** (Funnel lemma). *There exists a constant $c \geq 1$ so that the following holds. Let $d \geq 2$ be a constant. The size of a $k$-partitioner (excluding its output buffers) is bounded by $c \cdot k^{(d+1)/2}$. Assume $d \geq 2$ is such that $B^{1+\varepsilon} \leq M/3$ where $\varepsilon = 2/(d-1)$. Partitioning $N \geq k^d$ elements with FunnelPartition around $k-1$ pivots uses $N \lg(k)$ comparisons and incurs $O\big(\frac{N}{B}(\log_M(k)+1)+k\big)$ I/Os.*

**Algorithm 3** Operations on $k$-partitioners. The *full*() method returns whether a buffer has capacity for further elements. The *clear*() method removes all current elements from a buffer.

```
1:  procedure FunnelPartition(S[1..N], P[1..k − 1])
2:      Sort P
3:      Build k-partitioner, using P as pivots (assigned in in-order to nodes)
4:      r ← root node of k-partitioner
5:      Partition(r, S)
6:      Flush(r)
7:      Return k-partitioner output buffers

8:  procedure Partition(v, S[1..N])
9:      for x ∈ S do
10:         if x ≤ v.pivot then
11:             if v.leftBuffer.full() then
12:                 Partition(v.left, v.leftBuffer)
13:             v.leftBuffer.append(x)
14:         else
15:             if v.rightBuffer.full() then
16:                 Partition(v.right, v.rightBuffer)
17:             v.rightBuffer.append(x)
18:     S.clear()

19: procedure Flush(v)
20:     if v ≠ null then
21:         Partition(v.left, v.leftBuffer)
22:         Flush(v.left)
23:         Partition(v.right, v.rightBuffer)
24:         Flush(v.right)
```

**Proof.** Let $h = \lg(k)$; by definition of $k$, we have $h \in \mathbb{N}$. The space usage is given recursively by $s(k) = k' \cdot \lceil k^{d/2} \rceil + s(k') + k' \cdot s(k'')$; where $k' = 2^{\lceil h/2 \rceil}$ and $k'' = 2^{\lfloor h/2 \rfloor}$ are the number of leaves in the top funnel and the bottom funnels, respectively. Assuming $k = 2^{2^i}$ for $i \in \mathbb{N}$, this simplifies to $s(k) = k^{(d+1)/2} + (k^{1/2} + 1)s(k^{1/2})$, which satisfies $s(k) \leq S(k)$ where we set $S(k) = ck^{(d+1)/2}$ for a constant $c \geq 1$ that depends on initial conditions; if we use $s(4) = 2 \cdot 4^{d/2} + 3$ (space for the buffers and the 3 pivots), $c \geq 2.2$ suffices. The bound $s(k) \leq S(k)$ indeed remains valid even when $h$ is not a power of 2.

For the analysis of the I/O bound, let $M$ and $B$ with $B^{(d+1)/(d-1)} \leq M/3$ ("tall-cache assumption") be given. We follow the recursive construction of the funnel until a $\hat{k}$-partitioner $\hat{F}$ satisfies $S(\hat{k}) \leq M/3$, i.e., its it fits entirely into (a third of the) internal memory. For that choice, by the tall-cache assumption, the whole $\hat{k}$-partitioner and one block per child and parent buffer fit into internal memory: $S(\hat{k}) + (\hat{k} + 1)B \leq M$.

Call the edges/buffers connecting $\hat{F}$ to its parent and children *large* (if they exist). For the analysis, imagine removing all large edges; this leaves us with disconnected *base trees*, which in the $k$-partitioner are connected only by the large edges. Note that between any two levels, either none or all edges are large. However, unless $k = 2^{2^i}$, the height $\hat{h} = \lg(\hat{k})$ of a base tree can vary between $\underline{h} = \lg(M/(3c))/(d + 1)$ and $\overline{h} = 2\underline{h}$.

Now consider a call to PARTITION at the root of a base tree $\hat{F}$ with $\hat{k}$ leaves. Over the course of (recursively) pushing elements down through $\hat{F}$, we incur I/Os for loading the buffers and pivots of $\hat{F}$. Since base trees fit entirely into internal memory, unless there is another call triggered on a child base tree upon a full (large) buffer, we will only load buffers inside the $\hat{k}$-partitioner into memory once, at a total cost of $O(S(\hat{k})/B)$ I/Os; we also need to bring one block for each parent and child buffer into memory, using $O(\hat{k})$ I/Os. We now distinguish two cases.

Case (1): If $\hat{k} = k$, i.e., $\hat{F}$ is the entire $k$-partitioner. Since $\hat{F}$ (as well as one block per input and output buffer) fits into internal memory, we only need to load it once, at a cost of $O(S(k)/B + k)$ I/Os. This is $O(k^{(d+1)/2}/B + k) = O(k^d/B + k) = O(N/B + k) = O(N/B(\log_M(k) + 1) + k)$ as claimed.

Case (2): Otherwise, $\hat{k} < k$. We will argue that the following potential scheme pays for all I/Os costs: Whenever an element is inserted into a large buffer, it releases $\Theta(1/B)$ potential.

We first show that charging all elements for this released potential yields the desired bound. Between any two large buffers, one partitioning phase moves an element at least $\underline{h}$ levels down the tree. Overall, the $N$ elements need to travel at most $h = \lg(k)$ levels down, hence each element can be inserted at most $\lceil h/\underline{h} \rceil = O(\log_M(k) + 1)$ times into a large buffer, giving an overall potential of $O\big(\frac{N}{B}(\log_M(k) + 1)\big) = O\big(\frac{N}{B}(\log_M(k) + 1) + k\big)$ as claimed.

It remains to prove that the released potential exceeds the incurred I/O cost. First observe that we initially have to load each $\hat{F}$ when it is first used; likewise, we have to potentially load each $\hat{F}$ an additional time during the final FLUSH calls. These two load events sum to $O(S(k)/B + k)$ I/Os, which is $O(N/B + k)$ (as in case (1) above). Additionally, at any point in time during a PARTITION call on $\hat{F}$, we can get recursive PARTITION calls on $\hat{F}$'s child base trees in case their buffer becomes full; such a recursive call can evict $\hat{F}$ from the internal memory, and it has to be loaded before PARTITION resumes on $\hat{F}$, causing additional I/Os. We cannot say *when* these evictions will happen, but every eviction of $\hat{F}$ implies that a batch of $\beta$ elements have been pushed down from $\hat{F}$'s input buffer to a child's buffer, where $\beta$ is the size of the buffers below $\hat{F}$. By the recursive funnel construction, $\beta = \Omega(\hat{k}^d)$, so we must have seen a total release of $\Theta(\beta/B) = \Omega(\hat{k}^d/B)$ in potential. Since $\hat{k}$ is the first value with $S(\hat{k}) \leq M/3$, we have $S(2\hat{k}^2) > M/3$, which implies, $M = O(\hat{k}^{d+1})$. By the tall-cache assumption, this implies $B = O(M^{(d-1)/(d+1)}) = O\big((\hat{k}^{(d+1)})^{(d-1)/(d+1)}\big) = O(\hat{k}^{d-1})$, so $\hat{k} = O(\hat{k}^d/B)$. Hence, the cost of loading $\hat{F}$ again after an eviction of $O(S(\hat{k})/B + \hat{k}) = O(\hat{k}^d/B)$ I/Os is covered by a release in potential. ◀

On a conceptual level, a $k$-partitioner can be thought of as a cache-oblivious gadget for repeatedly partitioning with (variable) fan-out in $\Omega(M^{1/(d+1)}) \cap O(M^{2/(d+1)})$ that, when applied to $\Omega(M^{d/(d+1)})$ elements, uses $O(1/B)$ I/Os per element and partitioning round. This results from the bounds on $\hat{k}$ in base trees in the proof above.

## 4.3 Selecting Pivots

For funnelselect, we choose pivots $P_1, \ldots, P_{k-1}$ as follows; see also Figure 4: We first sample each element in the input $S$ with probability $p = 1/\lg N$. The resulting sample $\bar{S}$ is sorted. Finally we select the pivots as the ideal pivots in the sample, using the (expected) sample size $pN$: the $i$th pivot $P_i$ is the $\lfloor 1 + ipN/k \rfloor$th smallest element of $\bar{S}$. If $\bar{S}$ is too small, i.e., if $|\bar{S}| < \lfloor (k-1)pN/k \rfloor$ or $|\bar{S}| > 2pN$, we declare the pivot selection *failed* and repeat the sampling process.

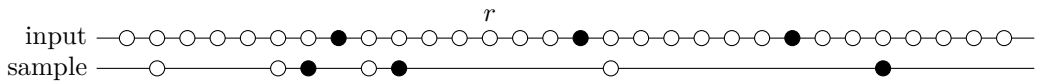



**Figure 4** Using sampling to select $k-1$ pivots for a $k$-way partitioning of the input, when $k=4$ and $N=30$. The expected sample size was 9 elements, but only 7 were actually sampled. Top shows the sorted input and $k-1$ ideal pivots, whereas the bottom shows similarly $k-1$ ideal pivots for a sample of the input points. Note the input of rank $r$ is in the 2nd block of white nodes defined by the ideal pivots, but would be in the 3rd block defined by the actual pivots from the sample.

In Lemma 6 we prove that all pivots are expected "close" to the ideal pivots in $S$. We call the $i$th pivot $P_i$ "$\xi$-bad" if its rank is more than $\xi$ away from its ideal rank, more formally

$$P_i \text{ is "}\xi\text{-bad"} \quad \text{if and only if} \quad \big|S \cap (-\infty, P_i]\big| \notin \big[\lfloor 1 + iN/k \rfloor - \xi\,,\, \lfloor 1 + iN/k \rfloor + \xi\big]\,.$$

We will call $P_i$ *bad* if it is $\xi$-bad; otherwise it is *good*. We call a pivot selection "bad" if any of the pivots is bad, and "good" otherwise. We select $\xi = \lceil N^{1/2+\delta} \rceil$ for a small constant $\delta$, where $0 < \delta < 1/6$. Since we assume $d \geq 3$, $k = \Theta\big(N^{1/d}\big)$ implies $N/k = \Omega\big(N^{2/3}\big)$, and $\xi < \frac{1}{2}N/k$ for sufficiently large $N$. We get the following fact:

▶ **Fact 4** (Good pivots). *If $\xi < \frac{1}{2}N/k$ and all pivots are good,*

**(a)** *no bucket $S_i$ contains more than $N/k + 2\xi \leq 2N/k$ elements, and*

**(b)** *a query for a rank $r$ will fall into one of at most two buckets: the $\lceil (r-\xi)/N \cdot k \rceil$th or the $\lceil (r+\xi)/N \cdot k \rceil$th bucket.*

Our analysis makes iterated use of the following Chernoff bound.

▶ **Lemma 5** ([10, Theorem 1.1]). *If $X_1, \ldots, X_n$ are independent random variables in $[0,1]$, and $X = X_1 + \cdots + X_n$, then for all $t > 0$ we have*

$$\Pr[X < \mathbb{E}[X] - t],\ \Pr[X > \mathbb{E}[X] + t] \quad \leq \quad e^{-2t^2/n}\,.$$

▶ **Lemma 6.** *The probability that $P_i$ is $\xi$-bad is bounded by $2\exp\big(-2\xi^2 p^2/N\big)$.*

**Proof.** $P_i$ is bad if its rank in $S$ is too small ("small bad") or too big ("big bad"). We first consider too small ranks. By choice, there are $ipN/k$ elements in $\bar{S}$ smaller than $P_i$; if $P_i$ has small-bad rank, all of these elements must be of rank $< iN/k - \xi$ in $S$. That means, from these $iN/k - \xi - 1$ smallest elements in $S$, we have chosen at least $ipN/k$ into $\bar{S}$. Since each choice is done independently with probability $p = 1/\lg N$, the number chosen for the sample is $X \stackrel{\mathcal{D}}{=} \mathrm{Bin}(iN/k - \xi - 1, p)$, a random variable with binomial distribution and expectation $\mathbb{E}[X] = p(iN/k - \xi - 1)$. We have

$$\begin{aligned}\mathbb{P}[P_i \text{ "small bad"}] \quad &\leq \quad \mathbb{P}[X \geq piN/k] \quad \leq \quad \mathbb{P}[X > \mathbb{E}[X] + p\xi] \\ &\underset{\text{Lemma 5}}{\leq} \exp\left(-2\frac{(p\xi)^2}{iN/k - \xi - 1}\right) \quad \leq \quad \exp\left(-\frac{2\xi^2 p^2}{N}\right).\end{aligned}$$

For the "big-bad" case, we must have chosen at most $ipN/k$ elements from the $iN/k + \xi + 1$ smallest elements in $S$ into $\bar{S}$. With $X' \stackrel{\mathcal{D}}{=} \mathrm{Bin}(iN/k + \xi + 1, p)$, we obtain

$$\mathbb{P}[P_i \text{ "big bad"}] \quad \leq \quad \mathbb{P}[X' \leq piN/k] \quad \leq \quad \mathbb{P}[X' < \mathbb{E}[X'] - p\xi] \quad \leq \quad \exp\left(-\frac{2\xi^2 p^2}{N}\right).$$

By the union bound, $P_i$ is bad with probability at most $2\exp\big(-2\xi^2 p^2/N\big)$. ◀

▶ **Lemma 7.** *The probability that the sample is too small to choose $k-1$ pivots is bounded by* $\exp\big(-2(pN/k-1)^2/N\big)$.

**Proof.** For the sample to be too small, we must have selected at most $(k-1)pN/k$ elements into the sample. Since $|\bar{S}| \stackrel{D}{=} \mathrm{Bin}(N,p)$, we have by Lemma 5

$$\mathbb{P}[\bar{S} \text{ too small}] \;=\; \mathbb{P}[|\bar{S}| \le \mathbb{E}[|\bar{S}|] - pN/k] \;\le\; \exp\left(-2\frac{(pN/k-1)^2}{N}\right). \qquad \blacktriangleleft$$

▶ **Corollary 8.** *With high probability, the pivot choice is well-defined and all $k-1$ pivots of one sampling round are good.*

**Proof.** By Lemmas 6 and 7 and the union bound, the probability that the sample is too small to choose our pivots or that any $P_i$ is bad is at most

$$2(k-1)\exp\big(-2\xi^2p^2/N\big) + \exp\big(-2(pN/k-1)^2/N\big) \;\le\; 2k\exp\big(-\Omega\big(N^{2\delta}/\lg^2 N\big)\big).$$

The inequality follows from $p = 1/\lg(N)$, $\xi = \Omega\big(N^{1/2+\delta}\big)$, $N/k = \Omega\big(N^{2/3}\big)$, and $\delta < 1/6$. This probability tends to 0 with speed superpolynomial in $N$. $\blacktriangleleft$

## 4.4 Truncated Partitioning

The algorithms from Section 3 achieve optimal cost from simply not recursing on subproblems not containing any query rank; after partitioning, it is obvious which subproblems are "query free". In a cache-oblivious algorithm, we have to truncate partitioning *inside* the $k$-partitioner.

After $k$-partitioning the input, elements are split into $k$ buckets; let us denote these buckets by $S_1, \ldots, S_k$. By Fact 4(b), when pivots are good, a query rank $r$ will fall in one of two buckets: one in $S(r) = \{S_{\lceil (r-\xi)/N\cdot k\rceil}, S_{\lceil (r+\xi)/N\cdot k\rceil}\}$. Buckets in the set $QF = \{S_1,\ldots,S_k\} \setminus \bigcup_{r\in R} S(r)$ do not contain any query ranks whenever pivots are good; we call these buckets *"expected query-free"*. Note that this is a property solely of $R$ and $k$ and hence $QF$ can be determined by scanning $R$ before partitioning commences.

When constructing the $k$-partitioner $F$, we check in a depth-first traversal whether all leaves below a binary partitioning node $v$ are in $QF$; if so, we remove $v$ and rewire its parent to send elements directly to an output buffer instead of $v$'s input buffer. The sizes of buffers between partitioning nodes and the Partition methods remain unchanged. By generating the output buffers for the leaves of $F$ consecutive in memory, before all internal buffers and reserving $2N/k$ space for each, this truncation operation simply changes one pointer.

## 4.5 Funnelselect

The overall algorithm FunnelSelect is shown in Algorithm 4. It applies one round of truncated $k$-partitioning as described above. For each of the resulting buckets, we then simply invoke an existing I/O-optimal cache-oblivious sorting algorithm and report the sought ranks. This can be done as a stack-based computation, so that no extra I/Os are paid for reporting elements, but instead they are reported while they are in main memory anyways from sorting.

# 5 Analysis

▶ **Theorem 9.** *Algorithm FunnelSelect is cache oblivious and uses $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os to report $q$ query ranks $r_1 < \cdots < r_q$ from an unsorted array of $N$ elements. With high probability it does not fail.*

**Algorithm 4** Our overall cache-oblivious multiple-selection algorithm.

```
1: procedure FunnelSelect(S[1..N], R[1..q], δ)
2:     p ← 1/lg N
3:     k ← 2^⌈lg(N)/d⌉
4:     ξ ← ⌈N^{1/2+δ}⌉
5:     Scan S, copy each element into the sample S̄ i.i.d. with prob. p
6:     if |S̄| ≤ (k−1)pN/k ∨ |S̄| < (1/2)Np ∨ |S̄| > 2Np then
7:         return Fail
8:     Sort S̄
9:     for i = 1, ..., k−1 do
10:        P_i ← S̄[⌊1 + ipN/k⌋] (select pivots from S̄)
11:    Construct k-partitioner F using pivots P_1, ..., P_{k−1}; let S_1, ..., S_k be its leaf buckets
12:    for r ∈ R do
13:        b_1 ← ⌈(r − ξ)/N · k⌉ and b_2 ← ⌈(r + ξ)/N · k⌉
14:        Mark leaf buckets S_{b_1} and S_{b_2} as expected query free
15:    for node v in bottom-up traversal of F do
16:        if both of v's children are marked expected query free then
17:            Mark v as expected query free
18:            Delete v's children
19:    for node v in preorder traversal of F do
20:        if v marked expected query free then
21:            Declare v's parent an expected query free output buffer
22:            Delete v
23:    Partition(F.root, S)
24:    Flush(F.root)
25:    L_1, ..., L_{k̂} ← leaf buckets in F
26:    ℓ ← 0
27:    for i = 1, ..., k̂ do
28:        R′ ← R ∩ (ℓ, ℓ + |L_i|]
29:        if R′ ≠ ∅ then
30:            if L_i marked expected query free ∨ |L_i| > 2N/k then
31:                return Fail
32:            else
33:                Sort L_i using an I/O-optimal cache-oblivious sorting algorithm.
34:                for r ∈ R′ do
35:                    Report L_i[r − ℓ]
36:        ℓ ← ℓ + |L_i|
```

**Proof.** Let us first deal with failures. We let the algorithm fail if $|\bar{S}|$ is smaller than $\frac{1}{2}Np$ or larger than $2Np$. From Lemma 5, with high probability, this does not happen. The only other cause for failure are bad pivots; by Corollary 8 with high probability, this also does not happen.

For the I/O cost, we consider the different steps in turn. The I/O cost for computing the pivots consists of $O(N/B)$ I/Os to scan the input to construct the sample $\bar{S}$ of size at most $\bar{N} = 2N/\lg N$. To sort the sample, we can use standard top-down mergesort, yielding $O(\bar{N}/B \cdot \lg(\bar{N}/M)) = O(N/B)$ I/Os for sorting $\bar{S}$ and $O(N/B)$ I/Os to extract the pivots from $\bar{S}$. In total selecting the pivots requires $O(N/B)$ I/Os.

The buffers of $F$ can be built sequentially, using $O(k^{(d+1)/2}/B) = O(N^{(d+1)/2d}/B) = O(N/B)$ I/Os (Lemma 3). Preparing the leaf buffers additionally touches $k = O(N^{1/d})$ positions; if $N^{1/d} > N/B$, then $B > N^{(d-1)/d}$ and by the tall-cache assumption, $M \geq B^{(d+1)/(d-1)} > N^{(d+1)/d}$, so the entire input fits in internal memory and the $k$ accesses are cached. The same applies for truncating the $k$-partitioner; anything touching $O(k)$ random positions incurs $O(N/B)$ I/Os. Marking leaves and nodes as expected query-free can be done by scanning $R$ (which is sorted), hence we use $O(q/B) = O(N/B)$ I/Os.

The key step is the $k$-partitioner. As in the proof of Lemma 3, we define $\hat{k}$ as the level in the recursive construction of the partitioner where the $\hat{k}$-partitioner first fits into internal memory. We overestimate the actual cost by always assuming the smallest $\hat{k} = (M/3c)^{1/(d+1)}$. As shown there, the $k$-partitioner has up to constant factors the same I/O cost as a repeated $\hat{k}$-way external-memory partitioning: $O(1/B)$ I/Os per element and $\hat{k}$-way split. This remains true when truncating the same subtrees in both algorithms. We can hence bound the cost of funnel partitioning as in Section 3 by charging the lengths of segments that are split further (white rectangles in Figure 1) to individual gaps $\Delta$. For MultiSelect$_{\mathrm{I/O}}$, charging a gap $\Delta$ on each level either $\Delta$ or the 2 segments containing its endpoints, whichever is less, was sufficient to cover all partitioning costs. For FunnelSelect, due to marking (up to) two leaf buckets as not query-free for the endpoints, we can have up to 4 segments on any level that still require partitioning. The rest of the analysis is the same, though, and with $C = O(1/B)$, $k = \hat{k}$, and $\alpha = 2/\hat{k}$, we obtain an upper bound of

$$C \cdot \left( \sum_{i=1}^{q+1} \Delta_i \log_{1/\alpha} \frac{4N}{\Delta_i} + \frac{N}{1-\alpha} \right) \quad = \quad O(\mathcal{B}_{\mathrm{I/O}} + N/B)$$

I/Os for partitioning.

The last part of the algorithm, solving subinstances of multiple selection within leaf buckets, could be solved recursively, but as we now show, fully sorting such buckets also fits our desired I/O bound. This improves the failure probability as sorting can be deterministic. A subproblem $L_i$ to recurse on is never declared query-free and hence moves all the $\lg k$ levels down the $k$-partitioner. For $N' = |L_i|$, $L_i$ hence contributes $\Theta\big(\frac{N'}{B} \log_M k\big) = \Theta\big(\frac{N'}{B} \log_{M/B} N\big)$ I/Os to the partitioning cost, since $k = \Theta\big(N^{1/d}\big)$ and under our tall-cache assumption $\lg \frac{M}{B} = \Theta(\lg M)$. This is an upper bound on the I/O cost of sorting $N'$ elements. Any I/O-efficient cache-oblivious sorting method (such as funnelsort) hence suffices for overall $O(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B})$ I/Os. ◀

▶ **Corollary 10.** *There exists a randomized cache-oblivious algorithm solving the multiple-selection problem using expected and with high probability $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ I/Os.*

**Proof.** FunnelSelect is formulated as a Monte-Carlo algorithm with worst-case time matching our expected-case time, but which can Fail occasionally. Repeating any failed execution turns it into a Las-Vegas algorithm with $O\big(\mathcal{B}_{\mathrm{I/O}} + \frac{N}{B}\big)$ expected I/Os; since the failure probability is superpolynomially small, we obtain the same bound with high probability. ◀

## 6 Partial Sorting

In internal memory, multiple selection would usually rearrange the input in place so that after the call to the multiple-selection algorithm, the sought elements are at indices $r_1, \ldots, r_q$. One would then not even return these elements explicitly. In external memory, this variant is less desirable, as one would have to pay $q$ I/Os for accessing the elements by index later. Hence

we defined the multiple-selection problem to return the elements of given ranks. However, it can be useful to also obtain the input partitioned around these returned values. Since all our multiple-selection algorithms conceptually follow an inorder traversal of a recursion tree and report sought elements (in sorted order) when they are identified, it is easy to augment the algorithms to produce a partitioned copy of the input array along the way. For funnelselect, we just have to make sure that output buckets are allocated sequentially in memory from left to right. That way we can output all elements falling between two returned pivots.

## 7 Allowing Identical Elements

In the previous sections we assumed elements to be distinct. A generic way to allow identical elements in the algorithms is by letting the algorithms process pairs $(x, i)$, where $x$ is the $i$th element in the array $S$, and break comparison ties between identical elements by comparing by their input position. This ensures all elements are considered distinct.

The drawback is that the computations need to process and store all these input positions. To avoid this overhead, one needs to address the problem directly by the individual algorithms. In the algorithms one needs to handle that multiple elements can be equal to the pivots. In the partitioning steps one needs to keep track of the number of elements equal to the pivots and only partition the elements not equal to pivots. Finally, one need to use this information gathered to handle that a pivot can span a range of ranks and be the answer to multiple query ranks.

## 8 Conclusion and Open Problems

We presented the first cache-oblivious multiple-selection algorithm that achieves the optimal I/O cost even when taking the (entropy of the) ranks to select into account.

A natural open problem is to find a *deterministic* cache-oblivious multiple-selection algorithm that achieves the same I/O bound as our randomized algorithm. Another interesting direction to explore is the "online" version of multiple selection studied in [3], where the ranks are given one after the other in arbitrary order and the algorithm has to produce the element of a given rank before the next rank is revealed. Investigating the practical performance of funnelselect is another route to pursue.

## References


1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.

2 Lars Arge, Mikael B. Knudsen, and Kirsten Larsen. A general lower bound on the I/O-complexity of comparison-based algorithms. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, Nicola Santoro, and Sue Whitesides, editors, *Algorithms and Data Structures, Third Workshop, WADS '93, Montréal, Canada, August 11-13, 1993, Proceedings*, volume 709 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 1993. doi:10.1007/3-540-57155-8_238.

3 Jérémy Barbay, Ankur Gupta, Srinivasa Rao Satti, and Jon Sorenson. Near-optimal online multiselection in internal and external memory. *Journal of Discrete Algorithms*, 36:3–17, January 2016. doi:10.1016/j.jda.2015.11.001.

4 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

5 Gerth Stølting Brodal and Rolf Fagerberg. Cache oblivious distribution sweeping. In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 426–438. Springer, 2002. doi:10.1007/3-540-45465-9_37.

6 Gerth Stølting Brodal and Rolf Fagerberg. On the limits of cache-obliviousness. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 307–315. ACM, 2003. doi:10.1145/780542.780589.

7 J. M. Chambers. Partial sorting [M1] (algorithm 410). *Commun. ACM*, 14(5):357–358, 1971. doi:10.1145/362588.362602.

8 David P. Dobkin and J. Ian Munro. Optimal time minimal space selection algorithms. *J. ACM*, 28(3):454–461, 1981. doi:10.1145/322261.322264.

9 Dorit Dor and Uri Zwick. Selecting the median. *SIAM Journal on Computing*, 28(5):1722–1758, 1999. doi:10.1137/s0097539795288611.

10 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

11 Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, March 1975. doi:10.1145/360680.360691.

12 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 285–298. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814600.

13 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012. doi:10.1145/2071379.2071383.

14 C. A. R. Hoare. Algorithm 63: partition. *Commun. ACM*, 4(7):321, 1961. doi:10.1145/366622.366642.

15 C. A. R. Hoare. Algorithm 64: quicksort. *Commun. ACM*, 4(7):321, 1961. doi:10.1145/366622.366644.

16 C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4(7):321–322, 1961. doi:10.1145/366622.366647.

17 Xiaocheng Hu, Yufei Tao, Yi Yang, and Shuigeng Zhou. Finding approximate partitions and splitters in external memory. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*. ACM, June 2014. doi:10.1145/2612669.2612691.

18 Kanela Kaligosi, Kurt Mehlhorn, J. Ian Munro, and Peter Sanders. Towards optimal multiple selection. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2005. doi:10.1007/11523468_9.

19 Helmut Prodinger. Multiple Quickselect – Hoare's Find algorithm for several elements. *Information Processing Letters*, 56(3):123–129, November 1995. doi:10.1016/0020-0190(95)00150-b.

20 Arnold Schönhage, Mike Paterson, and Nicholas Pippenger. Finding the median. *J. Comput. Syst. Sci.*, 13(2):184–199, 1976. doi:10.1016/S0022-0000(76)80029-3.

# Oriented Spanners

**Kevin Buchin**
Technical University of Dortmund, Germany

**Antonia Kalb**
Technical University of Dortmund, Germany

**Carolin Rehs**
Technical University of Dortmund, Germany

**Sampson Wong**
BARC, University of Copenhagen, Denmark

**Joachim Gudmundsson**
University of Sydney, Australia

**Aleksandr Popov**
Technical University of Eindhoven, The Netherlands

**André van Renssen**
University of Sydney, Australia

## Abstract

Given a point set $P$ in the Euclidean plane and a parameter $t$, we define an *oriented $t$-spanner* as an oriented subgraph of the complete bi-directed graph such that for every pair of points, the shortest cycle in $G$ through those points is at most a factor $t$ longer than the shortest oriented cycle in the complete bi-directed graph. We investigate the problem of computing sparse graphs with small oriented dilation.

As we can show that minimising oriented dilation for a given number of edges is NP-hard in the plane, we first consider one-dimensional point sets. While obtaining a 1-spanner in this setting is straightforward, already for five points such a spanner has no plane embedding with the leftmost and rightmost point on the outer face. This leads to restricting to oriented graphs with a one-page book embedding on the one-dimensional point set. For this case we present a dynamic program to compute the graph of minimum oriented dilation that runs in $\mathcal{O}(n^8)$ time for $n$ points, and a greedy algorithm that computes a 5-spanner in $\mathcal{O}(n \log n)$ time.

Expanding these results finally gives us a result for two-dimensional point sets: we prove that for convex point sets the greedy triangulation results in an oriented $\mathcal{O}(1)$-spanner.

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

**Keywords and phrases** computational geometry, spanner, oriented graph, greedy triangulation

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.26

**Related Version** *Full Version*: https://arxiv.org/abs/2306.17097

**Funding** *Aleksandr Popov*: Supported by the Dutch Research Council (NWO) under the project number 612.001.801.

## 1 Introduction

Computing geometric spanners is an extensively studied problem [5, 20]. Directed geometric spanners have also been considered [1]. Given a point set $P \subset \mathbb{R}^d$ and a parameter $t$, a *directed $t$-spanner* $G = (P, E)$ is a subgraph of the complete bi-directed geometric graph on $P$ such that for every pair of points $p, p'$, the shortest path in $G$ is at most a factor $t$ longer than the shortest path in the complete graph, that is, $|p - p'|$. The *dilation* of $G$ then is the smallest such $t$. Formally, $t = \max\left\{ \frac{|d_G(p,p')|}{|p-p'|} \;\middle|\; p, p' \in P \right\}$, where $d_G(p, p')$ denotes the *shortest path from $p$ to $p'$ in $G$*.

(Directed) geometric spanners have a wide range of applications, ranging from wireless ad-hoc networks [7, 21] to robot motion planning [12] and the analysis of road networks [2, 14]. In all of these applications one might want to avoid adding the edge $(v, u)$ if the edge $(u, v)$ was included: in ad-hoc networks this may reduce interference, in motion planning it may reduce congestion and simplify collision avoidance, in road networks this corresponds to one-way





roads or tracks, which may be necessary because of space limitations, and in communication networks one could require two neighbouring devices not to exchange data by the same (bi-directional) direct connection, for example, in two-way authentication.

This motivates our study of *oriented graphs* as spanners, i.e. directed spanners $G = (P, E)$ where $(p, p') \in E$ implies $(p', p) \notin E$. With this restriction, if the edge $(p, p')$ is added, the dilation in the other direction is never 1. Even worse, given a set $P$ of three points, where $p$ and $p'$ are very close to each other and $p''$ is far away from both, any oriented graph will have arbitrarily high dilation for either $(p, p')$ or $(p', p)$ (see Figure 1). Therefore, considering the dilation for an oriented graph as $t = \max\{\frac{|d_G(p,p')|}{|p-p'|} \mid p, p' \in P\}$ would not tell us much about the quality of the spanner. To obtain meaningful results, we define *oriented dilation*.

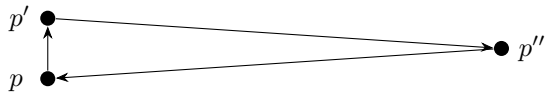



**Figure 1** If $p$ and $p'$ are very close to each other and $p''$ is far away from both, any oriented graph will have arbitrarily high (directed) dilation.

By $C_G(p, p')$ we denote the *shortest oriented cycle* containing the points $p$ and $p'$ in an oriented graph $G$. The *optimal oriented cycle* $\Delta(p, p')$ for two points $p, p' \in P$ is the shortest oriented cycle containing $p$ and $p'$ in the complete graph on $P$. Notice, $\Delta(p, p')$ is the triangle $\Delta_{pp'p''}$ with $p'' = \underset{p^* \in P \setminus \{p,p'\}}{\arg\min} \big(|p - p^*| + |p^* - p'|\big)$.

▶ **Definition 1** (oriented dilation). *Given a point set $P$ and an oriented graph $G$ on $P$, the* oriented dilation of two points $p, p' \in P$ *is defined as*

$$\mathrm{odil}(p, p') = \frac{|C_G(p, p')|}{|\Delta(p, p')|}.$$

*The dilation $t$ of an oriented graph is defined as $t = \max\{\mathrm{odil}(p, p') \mid \forall p, p' \in P\}$.*

An oriented graph with dilation at most $t$ is called an *oriented $t$-spanner*. We frequently contrast our results to known results on undirected geometric spanners, and refer to the known results by using the adjective *undirected*. Our new measure for oriented graphs is similar to the definition of dilation in round trip spanners [9, 10] that has been considered in the setting of (non-geometric) directed graph spanners; but round trip spanners require a starting graph, and using the complete bi-directed geometric graph would not give meaningful results.

In this paper, we initiate the study of oriented spanners. As is common for spanners, our general goal is to obtain *sparse* spanners, i.e. with linear number of edges. The goal can be achieved by bounding the number of edges explicitly or by restricting to a class of sparse graphs like plane graphs [5]. We refer to a spanner as a *minimum (oriented) spanner* if it minimises $t$ under the given restriction.

It is known that computing a minimum undirected spanner with at most $n - 1$ edges, i.e. a minimum dilation tree, is NP-hard [15]. The corresponding question for oriented spanners asks for the *minimum dilation cycle*. We prove this problem to be NP-hard in Section 3.1.

The problem of computing the minimum undirected spanner restricted to the class of plane straight-line graphs is called the *minimum dilation triangulation* problem; its hardness is still open [14, 15], but it is conjectured to be NP-hard [6]. As this undirected problem can be emulated in the oriented setting by suitable vertex gadgets, it is unlikely that finding a *minimum plane straight-line (oriented) spanner* can be done efficiently.

Therefore, in Section 2, we start with one-dimensional point sets. For such a point set $P$ with $n$ points, we can give a minimum spanner with $3n-6$ edges. However, if we are interested in a one-dimensional result analogous to minimum plane spanners, this spanner is not suitable: it has no plane embedding with leftmost and rightmost point on the outer face. Therefore, we restrict our attention to a graph class that is closer to the plane case for two-dimensional point sets: one-page book embeddings.

We show how to compute a $t$-spanner which is a one-page plane book embedding for a one-dimensional point set in Section 2.2. We prove that with a greedy algorithm, we can always generate such a $t$-spanner with $t \leq 5$ in $\mathcal{O}(n \log n)$ time. An optimal one-page plane spanner can be computed in $\mathcal{O}(n^8)$ time (Theorem 11).

As the resulting spanner is outerplanar, this particular class of graphs is also motivated by the problem of finding a minimum plane spanner for points in convex position. Using these results, in Section 3.2, we show that suitably orienting the greedy triangulation leads to oriented $\mathcal{O}(1)$-spanners for two-dimensional point sets in convex position (Theorem 16). For general (non-convex) point sets, there are examples where all orientations of the greedy triangulation have large dilation.

The greedy triangulation fulfils the $\alpha$-diamond-property [11], and all triangulations with this property are undirected $\mathcal{O}(1)$-spanners. This raises the question whether all triangulations fulfilling this property are also oriented $\mathcal{O}(1)$-spanners for convex point sets. In Section 3.3 we answer this question negatively.

**Table 1** Overview of the results of the paper.

| Point set | Spanner type | Dilation | Time complexity | Reference |
|---|---|---|---|---|
| 2-dim. | unrestricted | minimum | NP-hard | Theorem 14 |
| 2-dim. | unrestricted | $\leq 2$ | $\mathcal{O}(n^2)$ | Proposition 13 |
| 2-dim. | plane | minimum | Min.Dil. Triangulation | Observation 15 |
| 2-dim. convex | plane | $\mathcal{O}(1)$ | $\mathcal{O}(n \log n)$ | Theorem 16 |
| 1-dim. | unrestricted | 1 | $\mathcal{O}(n)$ | Corollary 3 |
| 1-dim. | 2-page-plane | $\leq 2$ | $\mathcal{O}(n)$ | Proposition 4 |
| 1-dim. | 1-page-plane | minimum | $\mathcal{O}(n^8)$ | Theorem 11 |
| 1-dim. | 1-page-plane | $\leq 5$ | $\mathcal{O}(n \log n)$ | Theorem 10 |

## 2 One-Dimensional Point Sets

We first focus on points in one dimension. We will always draw points on a horizontal line with the minimum point leftmost, and the maximum point rightmost. We observe that in one dimension only the dilation of a linear number of candidate pairs needs to be checked.

▶ **Lemma*** **2.** *Let $P$ be a one-dimensional point set of $n$ points. The oriented dilation $t$ of an oriented graph $G$ on $P$ is*

$$t = \max\{\mathrm{odil}(p_i, p_{i+2}), \mathrm{odil}(p_j, p_{j+3}) \mid 1 \leq i \leq n-2, 1 \leq j \leq n-3\}.$$

This observation directly leads to an oriented 1-spanner with $3n-6$ edges for every one-dimensional point set.

The proofs of results marked by $*$ are given in a long version of this paper.

▶ **Corollary 3** (oriented 1-spanner). *For every one-dimensional point set $P$, $G = (P, E)$ with*

$$E = \{(p_i, p_{i+1}), (p_{j+2}, p_j), (p_{k+3}, p_k) \mid 1 \leq i \leq n-1, 1 \leq j \leq n-2, 1 \leq k \leq n-3\}$$

*is an oriented 1-spanner for $P$ (see Figure 2).*

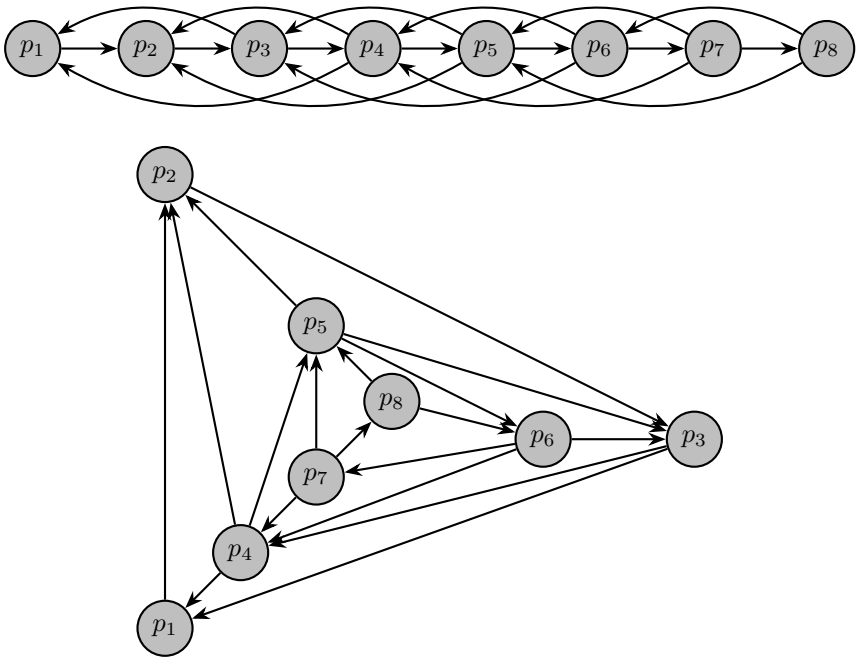



**Figure 2** One-dimensional oriented 1-spanner and its plane embedding in the Euclidean space.

In two dimensions, plane (straight-line) spanners are of particular interest. The natural one-dimensional analogue to plane straight-line graphs are one- and two-page book embeddings [3, 8, 13, 22].

A *one-page book embedding* of a graph corresponds to an embedding of the vertices as points on a line with the edges drawn without crossings as circular arcs above the line. In a *two-page book embedding* an edge may be drawn as an arc above or below the line. In such a (one- or two-page) book embedding, for consecutive points on the line, we may draw their edge straight on the line. We call a graph *one-page plane* (respectively *two-page plane*) if it has a one-page (respectively two-page) book embedding.

In particular, one-page plane graphs are outerplanar graphs and correspond to plane straight-line graphs if we embed the points on a (slightly) convex arc instead of on a line. Two-page plane graphs are a subclass of planar graphs, while any planar graph has a four-page book embedding [22].

As we argue next, the 1-spanner of Corollary 3 is not two-page plane (and therefore also not one-page plane). This follows from a stronger statement: the graph has no plane embedding with the first and last point on the outer face. Suppose the graph would have such an embedding. The graph has $3n - 6$ edges, but no edge between the first and the last point for $n > 4$. Thus, we could add this edge while maintaining planarity, which contradicts the fact that a planar graph has at most $3n - 6$ edges. Interestingly, the 1-spanner is planar. We construct a stack triangulation by adding points from left to right. The first three points form a triangle. Then, we inductively add the next point into the triangle formed by the previous three points. See Figure 2.

## 2.1 Two-Page Plane Spanners

As argued above, the 1-spanner given in Corollary 3 is not two-page plane (and thus not one-page plane). Moreover, by Lemma 2, no two-page plane 1-spanner can exist. However, we can give a two-page plane 2-spanner for every one-dimensional point set:

▶ **Proposition*** **4** (two-page plane 2-spanner)**.** *For every one-dimensional point set $P$, the graph $G = (P, E)$ with $E = \{(p_i, p_{i+1}), (p_{j+2}, p_j) \mid 1 \leq i \leq n-1, 1 \leq j \leq n-2\}$ (see Figure 3) is a two-page plane oriented 2-spanner for $P$.*

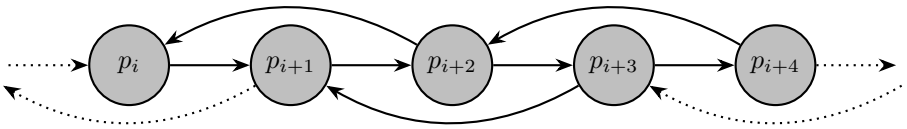



**Figure 3** Part of $G = (P, E)$ with $E = \{(p_i, p_{i+1}), (p_{j+2}, p_j) \mid 1 \leq i \leq n-1, 1 \leq j \leq n-2\}$.

## 2.2 One-Page Plane Spanners

The 2-spanner given by Proposition 4 is two-page plane, but not one-page plane. As noted above, one-page plane graphs on one-dimensional point sets correspond to plane straight-line graphs if we interpret the point set as being convex. We thus place particular focus on one-plane plane graphs, since they are not only of interest in their own right but also aid us in finding oriented plane spanners in two-dimensions.

By *maximal one-page plane*, we mean a one-page plane graph $G = (P, E)$ such that for every edge $e \notin E$, the graph $G' = (P, E \cup \{e\})$ is not one-page plane. We call the edge set $\{(p_i, p_{i+1}) \mid 1 \leq i \leq n-1\}$ the *baseline*. A directed edge $(p_j, p_i)$ with $i < j$ is a *back edge*.

We refer to an oriented one-page plane graph that includes a baseline and all other edges are back edges as a one-page-plane-baseline graph, for short 1-*PPB graph*. Lemma 5 shows that a 1-PPB graph has smaller dilation than an oriented graph with the same edge set but another orientation. Without loss of generality, we consider only 1-PPB graphs instead of one-page plane graphs in general.

▶ **Lemma*** **5.** *Let $G = (P, E)$ be a one-page plane oriented $t$-spanner for a one-dimensional point set $P$. Let $E'$ be an edge set where the edges are incident to the same vertices as $E$, but the orientation may be different. If we orientate such that $G' = (P, E')$ is a 1-PPB graph, then the dilation of $G'$ is at most $t$.*

▶ **Lemma*** **6.** *Let $P$ be a one-dimensional point set of $n$ points. The oriented dilation $t$ of a 1-PPB graph for $P$ is*

$$t = \max\{\mathrm{odil}(p_i, p_{i+2}) \mid 1 \leq i \leq n-2\}.$$

Lemma 6 holds for every 1-PPB graph, even if it is not maximal. Proposition 4 shows that Lemma 6 does not hold for non-one-page plane graphs.

Due to one-page planarity, for a point set $P$ with $|P| > 3$, every graph $G$ contains a tuple $p_i, p_{i+2} \in P$ where $C_G(p_i, p_{i+2}) \neq \Delta(p_i, p_{i+2})$. Combining this with Lemma 6, we get:

▶ **Corollary 7.** *There is no one-page plane 1-spanner for any point set $P$ with $|P| > 3$.*

However, there are point sets with a one-page plane almost 1-spanner. Let $G = (P, E)$ be a graph with $|P| = 5$, $E = \{(p_i, p_{i+1}) \mid 1 \leq i \leq n-1\} \cup \{(p_3, p_1), (p_5, p_3)\}$ and the distances $p_2 - p_1 = p_5 - p_4 = \varepsilon$ and $p_3 - p_2 = p_4 - p_3 = 1$ (see Figure 4). It holds that $t = \text{odil}(p_2, p_4) = \frac{2+2\varepsilon}{2}$. For an arbitrary small $\varepsilon$, $G$ is a one-page plane almost 1-spanner.

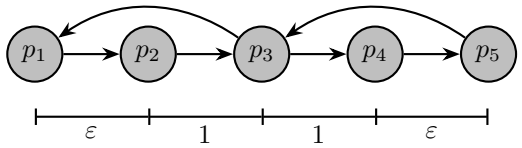



**Figure 4** 1-PPB almost 1-spanner.

▶ **Observation*** **8.** *There are one-dimensional point sets where no one-page plane oriented $t$-spanner exists for $t < 2$.*

We construct a 1-PPB spanner by starting with the base line and greedily adding back edges sorted by length from shortest to longest if they do not cross any of the edges already added. The following leads to a simple, greedy algorithm for constructing this graph: The first edge that we need to add is the shortest edge between two points with exactly one point in between. Imagine deleting the point that was in between. Then again, we need to add the shortest edge with exactly one point in between and so on, until only two points are left. By maintaining the points in a linked list and the relevant distances in a priority queue, this leads to a run time of $\mathcal{O}(n \log n)$.

To bound the dilation of the resulting graph, we need the concept of a *blocker*.

▶ **Definition 9** (blocker). *Let $E$ be the greedily computed back edge set. Because the resulted graph $G = (P, E \cup \{(p_i, p_{i+1}) \mid 1 \leq i \leq n-1\})$ is maximal, $(p_j, p_i) \notin E$ for $i + 2 \leq j$ implies there is a shorter edge in $E$ which intersects with $(p_j, p_i)$. (The greedy algorithm added this edge first and discarded $(p_j, p_i)$ in a later iteration of the loop.) For the shortest edge intersecting $(p_j, p_i)$, we say it* blocks *$(p_j, p_i)$. The edge can* be blocked by *$(p_k, p_m)$ with $k > j$ and $i < m < j$ or $(p_m, p_{k'})$ with $k' < i$ and $i < m < j$ (see Figure 5).*

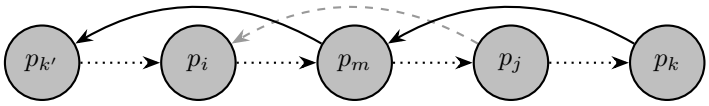



**Figure 5** $(p_j, p_i)$ can be blocked by $(p_k, p_m)$ or $(p_m, p_{k'})$ with $i < m < j$, $k > j$ and $k' < i$.

▶ **Theorem*** **10** (one-page plane 5-spanner). *Given a one-dimensional point set $P$ of size $n$, a one-page plane oriented 5-spanner can be constructed in $\mathcal{O}(n \log n)$ time.*

The full proof is given in the full version, but, since the proof of correctness for the greedy two-dimensional algorithm (see Section 3.2) works similarly, we give a proof sketch here.

**Proof sketch.** Due to Lemma 6, it is sufficient to bound the dilation $\text{odil}(p_i, p_{i+2})$ of tuples $p_i, p_{i+2}$ with one point in-between. Let $(b_1, p_{i+1})$ be the blocker of the optimal edge $(p_{i+2}, p_i)$ (blue). There could be a "sequence of blockers" $b_1, \ldots, b_{j+1}$, such that $(b_k, p_{i+1})$ blocks $(b_{k-1}, p_i)$ for $2 \leq k \leq j + 1$ (green), see Figure 6.

Combining this with a lower bound on the distance of a tuple $b_{i-2}, b_i$ for $3 \leq i \leq j+1$ (violet), we show that there are only two blockers $b_j$ and $b_{j+1}$, whose the distance to $p_{i+1}$ is larger than $p_{i+1} - p_i$. There are two cases for the shortest oriented cycle $C_G(p_i, p_{i+2})$. For $(b_{j+1}, p_i) \in E$ holds $\operatorname{odil}(p_i, p_{i+2}) = 4$. The worst-case is that $(b_{j+1}, p_i) \notin E$ is blocked by some edge $(p_{i+1}, p_l)$ with $l < i$ (red). Then the dilation is

$$\operatorname{odil}(p_i, p_{i+2}) = \frac{(b_1 - p_l) \cdot 2}{(p_{i+2} - p_i) \cdot 2} \leq \frac{b_1 - p_{i+1} + p_{i+1} - p_l}{p_{i+2} - p_i} \leq \frac{p_{i+2} - p_i + 4 \cdot (p_{i+1} - p_i)}{p_{i+2} - p_i} = 5.$$

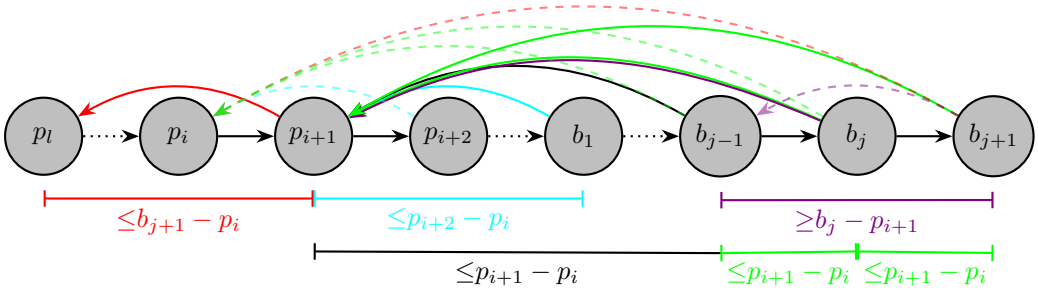



**Figure 6** Blockers in the proof of Theorem 10.

Figure 7 shows a point set $P$ and its greedily constructed spanner $G$. The oriented dilation of $G$ is $t = \frac{5-7\varepsilon}{1+\varepsilon}$. Thus, $G$ is a 5-spanner for $P$.

However, Figure 8 shows a $t$-spanner with $t = \frac{2-2\varepsilon}{1+\varepsilon} < 2$ for the same point set. Therefore, the greedy algorithm does not return the minimum spanner for every given one-dimensional point set.

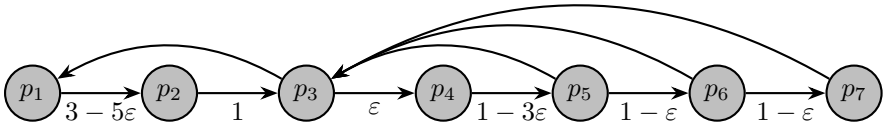



**Figure 7** Example of a greedily constructed spanner, its dilation is $t = \operatorname{odil}(p_2, p_4) = \frac{5-7\varepsilon}{1+\varepsilon} < 5$.

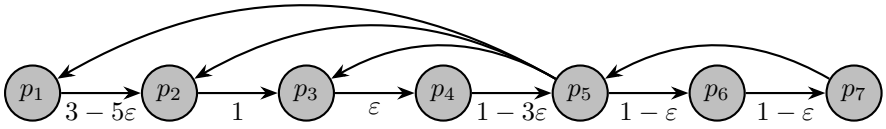



**Figure 8** A spanner on the point set of Figure 7 with dilation $t = \operatorname{odil}(p_2, p_4) = \frac{2-2\varepsilon}{1+\varepsilon} < 2$.

The minimum dilation of a 1-PPB spanner for a one-dimensional point set is larger than 1 (Observation 8), unlike non-plane oriented spanners (Corollary 3). Algorithm 1 computes in $\mathcal{O}(n^8)$ time a minimum 1-PPB spanner for a given point set. The dynamic program is based on the following idea. By Lemma 6, it holds that $t = \max\{t', t'', \operatorname{odil}(p_{k-1}, p_{k+1})\}$, where $t'$ is the minimum dilation for $\{p_1, \ldots, p_k\}$ and $t''$ is the minimum dilation for $\{p_k, \ldots, p_n\}$. Due to one-page-planarity, if $(p_r, p_l) \in E$, it holds that $(p_j, p_i) \notin E$ for $l < i < r < j$ and $i < l < j < r$. We test all candidates for a split point $p_k$. By adding the edges $(p_n, p_k)$

and $(p_k, p_1)$, we can compute the optimal edge set for $\{p_1, \ldots, p_k\}$ and $\{p_k, \ldots, p_n\}$ (almost) independently. (We need the parameters $l'$ and $r'$ which represent the only edges needed to consider $\text{odil}(p_{k-1}, p_{k+1})$.)

▶ **Theorem*** **11** (Optimal 1-PPB). *Given a one-dimensional point set $P$ of size $n$, the minimum one-page plane oriented spanner for $P$ can be calculated in $\mathcal{O}(n^8)$ time.*

**Algorithm 1** Minimum One-Page Plane Spanner.

**Require:** one-dimensional point set $P = \{p_1, \ldots, p_n\}$ (numbered from left to right)
**Ensure:** minimum one-page plane oriented spanner for $P$

Initialise table $\text{OE} = [1, \ldots, n] \times [1, \ldots, n] \times [1, \ldots, n] \times [1, \ldots, n]$
*// ODIL($E'$) returns dilation of $G = (P, E' \cup \{(p_i, p_{i+1}) \mid 1 \leq i \leq n-1\})$*
*// OE($l, l', r', r$)= back edges set $E'$ of a minimum spanner $G$ for $\{p_l, \ldots, p_r\}$ with $(p_{l'}, p_l) \in C_G(p_l, p_{l+1})$ and $(p_r, p_{r'}) \in C_G(p_{r-1}, p_r)$*
Fill table by dynamic program based on the recursion formula:

$$\text{OE}(l, l', r', r) = \begin{cases} \text{“invalid”, if } l' < l+2,\ r' > r-2 \text{ or } l > r \text{ (contradicts definition)} & \text{(i)} \\ \text{“invalid”, if } r < l+2 \text{ (no oriented spanners for } |P| < 3) & \text{(ii)} \\ (p_r, p_l), \text{ if } l' = r \text{ and } r' = l \text{ (spanner for } |P| = 3 \text{ is a cycle)} & \text{(iii)} \\ \text{“invalid”, if } l' < r,\ r' > l \text{ and } l' > r' \text{ (contradicts planarity)} & \text{(iv)} \\ E' = \text{OE}(l, l', k_r, r-1) \cup \{(p_r, p_l)\}, \text{ if } l' < r \text{ and } r' = l, \text{ choose } l \leq k_r \leq r-3 \text{ s.t. } \text{ODIL}(E') \text{ is minimal} & \text{(v)} \\ E' = \text{OE}(l+1, k_l, r', r) \cup \{(p_r, p_l)\}, \text{ if } l' = r \text{ and } r' > l, \text{ choose } l+3 \leq k_l \leq r \text{ s.t. } \text{ODIL}(E') \text{ is minimal} & \text{(vi)} \\ E' = \text{OE}(l, l', k_r, k) \cup \text{OE}(k, k_l, r', r) \cup \{(p_r, p_l)\}, \text{ if } l' < r,\ r' > l \text{ and } l' \leq r', \text{ choose } l \leq k_r \leq k-2,\ l+2 \leq k \leq r-2 \text{ and } k+2 \leq k_l \leq r \text{ s.t. } \text{ODIL}(E') \text{ is minimal} & \text{(vii)} \end{cases}$$

$E' \leftarrow \text{OE}(1, l', r', n)$, choose $l'$ and $r'$ s.t. $\text{ODIL}(E')$ is minimal
**return** $G = (P, E' \cup \{(p_i, p_{i+1}) \mid 1 \leq i \leq n-1\})$ *// add baseline*

## 3 Two-Dimensional Point Sets

In the one-dimensional case, we have seen that a 1-spanner exists for every point set, though it is not plane. In two dimensions, the complete bi-directed graph is always a directed 1-spanner. This is not the case for oriented dilation. There we have point sets where no 1-spanner exists, even more, no 1.46-spanner exists:

▶ **Observation*** **12.** *There are two-dimensional point sets for which no oriented $t$-spanner exists for $t < 2\sqrt{3} - 2 \approx 1.46$.*

While an oriented complete graph does not lead to a 1-spanner, it does yield a 2-spanner:

▶ **Proposition*** **13.** *For every point set an oriented 2-spanner can be constructed by orienting a complete graph.*

## 3.1 Hardness

We now have shown that although a 1-spanner does not exist for every two-dimensional point set, we get a 2-spanner via the oriented complete graph. However, this leads to a graph with $\frac{1}{2}n \cdot (n-1)$ edges, so this graph is neither sparse nor plane.

We will see that computing a sparse minimum oriented spanner is NP-hard:

▶ **Theorem*** **14.** *Given a two-dimensional set $P$ of $n$ points and the parameters $t$ and $m'$, it is NP-hard to decide if there is an oriented $t$-spanner $G = (P, E)$ with $|E| \leq m'$.*

For computing a minimum *plane* oriented spanner, i.e. a minimum oriented dilation triangulation, hardness remains open. In the undirected setting this question is a long-standing open problem [6, 14, 15]. To our knowledge, it is not even known whether a PTAS for this problem exists, and for several related open problems there is no FPTAS [15], unless $P = NP$. We show the following relative hardness result.

▶ **Observation 15.** *An FPTAS for minimum plane oriented spanner would imply an FPTAS for minimum dilation triangulation.*

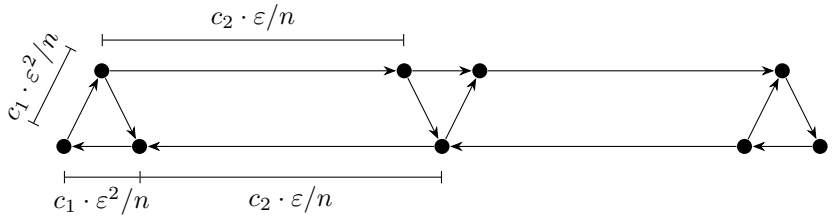



**Figure 9** By replacing each point with this construction, we can reduce the minimum dilation triangulation problem to the minimum plane oriented spanner problem.

We sketch a reduction from the minimum dilation triangulation problem. Suppose we are given a set $P$ of $n$ points. The idea is to replace every point with the gadget depicted in Figure 9. The gadget consists of $3 \cdot \lceil \frac{4n}{3} \rceil$ points, positioned along a line in small triangles. Since the triangles are small relative to the distance between the triangles, we obtain an oriented dilation of less than $(1 + c\varepsilon)$ within the gadget by connecting the gadget points as in the figure, where $c > 0$ is a constant that we can choose by suitably picking $c_1$ and $c_2$.

The gadget leaves us with room for two edges to every one of $n$ points, one in each direction, without disturbing planarity. Therefore, minimising the oriented dilation in this setting while requiring planarity should pick the edges that correspond to those in the minimum dilation triangulation, except if they are $\varepsilon$-close.

## 3.2 Greedy Triangulation

Given the relative hardness of computing minimum plane oriented spanners, we next investigate the question of whether plane oriented $\mathcal{O}(1)$-spanners exist. In the undirected setting, prominent examples for plane constant dilation spanners are the Delaunay triangulation and the greedy triangulation. Our main result on oriented spanners in two dimensions is that the greedy triangulation is an $\mathcal{O}(1)$-spanner for convex point sets (i.e. point sets for which the points lay in convex position).

The *greedy triangulation* of a point set is the triangulation obtained by considering all pairs of points by increasing distance, and by adding the straight-line edge if it does not intersect any of the edges already added. The greedy triangulation can be computed in linear time from the Delaunay triangulation [18], and in linear time for a convex point set if the order of the points along the convex hull is given [19].

It is known that the dilation of the undirected greedy triangulation is bounded by a constant. This follows from the fact that the greedy triangulation fulfils the $\alpha$-diamond-property [11]. The currently best upper bound on the dilation $t$ of such a triangulation is $t \leq \frac{8(\pi-\alpha)^2}{\alpha^2 \sin^2(\alpha/4)}$ with a lower bound on $\alpha$ of $\pi/6$ for the greedy triangulation [4].

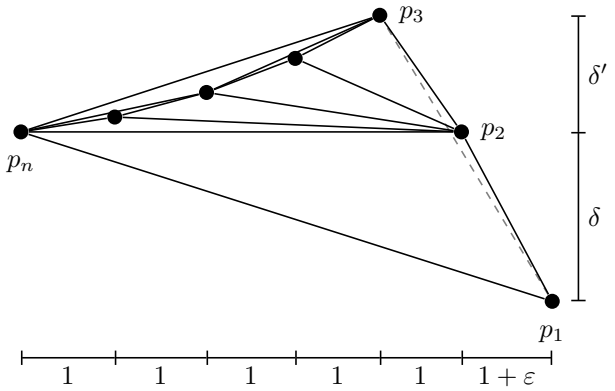



**Figure 10** Greedy triangulation for a non-convex point set. It is is also a minimum weight triangulation.

We first observe that restricting to convex point sets is necessary to obtain constant dilation from orienting the greedy triangulation. For this, consider the greedy triangulation $T = (P, E)$ on the following non-convex point set $P = \{p_1, \ldots, p_n\}$. As shown in Figure 10, we place the points $\{p_3, \ldots, p_n\}$ on a very flat parabola. By arbitrarily decreasing the $y$-distances $\delta, \delta' > 0$, we reduce the construction to an almost one-dimensional problem. The $x$-distance of each consecutive point pair $p_i, p_{i+1}$ is 1 for $2 \leq i < n$. We place $p_1$ slightly right of $p_2$ such that the $x$-distance is $1 + \varepsilon$ for a small $\varepsilon > 0$. Therefore, the $x$-distance of $p_1$ to $p_i$ is $\varepsilon$-larger than $x$-distance of $p_2$ to $p_{i+1}$ for $3 \leq i < n$. By this, the greedy triangulation added $(p_2, p_{i+1}) \in E$ and discarded $(p_1, p_i) \notin E$ for $3 \leq i < n$. Further, we place $p_1$ slightly below $p_2$ so that, due to planarity, $p_1$ is only adjacent to $p_2$ and $p_n$. Since $p_1$ has degree 2, for any orientation of $T$, every shortest oriented cycle containing $p_1$ contains the subpath $p_n, p_1, p_2$ or vice versa. From this construction, it follows that $\mathrm{odil}(p_1, p_3) \geq \frac{n+\varepsilon}{2+\varepsilon}$, regardless of the orientation of $T$. Therefore, every orientation of greedy triangulation has oriented dilation $\Omega(n)$.

In contrast, for convex point sets, the greedy triangulation results in a $\mathcal{O}(1)$-spanner. For this, we use a *consistent* orientation of the edges. This means each face of an oriented triangulation is confined by an oriented cycle. If such an orientation exists for a given triangulation, we call it an *orientable triangulation*. Since the dual graph of a triangulation of a convex point set is a tree, it is orientable.

▶ **Theorem 16.** *By orienting the greedy triangulation of a convex two-dimensional point set $P$ consistently, we get a plane oriented $\mathcal{O}(1)$-spanner for $P$.*

**Proof.** Let $T = (P, E)$ be the greedy triangulation of $P$ and $G = (P, \overrightarrow{E})$ its consistent orientation (which is unique up to reversing all edges). Note that $T$ is an undirected graph, whereas $G$ is directed. To improve readability, undirected edges are written with curly

brackets and directed edges with round brackets. Due to $\alpha$-diamond property, the undirected dilation of any greedy triangulation can be bounded by a constant. Let $t_g$ be the (smallest such) constant.

We distinguish between i) $\{p, p'\}$ is in $E$ or ii) not.

For i), we prove that there is a path from $p$ to $p'$ in $T - \{p, p'\}$ of length in $\mathcal{O}(|\Delta(p, p')|)$, where $\Delta(p, p')$ is a smallest triangle in the complete graph incident to $p$ and $p'$. By that, we bound the length of the shortest oriented cycle $C_G(p, p')$.

Let $q \in P$ be the third point incident to $\Delta(p, p')$. Let $\Pi$ be the undirected path from $p$ to $q$ in $T$ with first edge $\{p, q'\}$, w.l.o.g. $q' \neq p'$. (If $q' = p'$, then the proof would be the same with switched roles for $p$ and $p'$.)

If $\{q', p'\} \in E$, then $\Delta_{pq'p'} \in E$. Regardless of the orientation of $\{p, q'\}$, we can bound $|C_G(p, p')| \leq |\Delta_{pq'p'}| \leq 2 \cdot t_g \cdot |\Delta(p, p')|$. However, $\{q', p'\}$ could be blocked by another edge. Since $P$ is convex and $T$ is planar, this edge is incident to $p$. Analogous to Theorem 10, we show that there could be a "sequence of blockers" (see Definition 9), but that there is path from $p$ to $q$ of length $\mathcal{O}(|p - q|)$.

Let $b_{j+2c}$ be such that $\{q', b_{j+2c}\} \in E$ and $\{q', b_{j+2c}\}$ blocks $\{q', p'\}$. Let the points $b_1, \ldots, b_{j+2c}$ be ordered such that $\{p, b_i\} \in E$ is the shortest edge that blocks $\{q', b_{i-1}\}$ with $b_0 = p'$ (see Figure 11a).

The following inequalities hold for these blockers:

$$|p - b_i| \leq |q' - b_{i-1}|, \tag{1}$$

$$|p - b_i| \leq |p - b_k| \text{ for } 1 \leq i < k \leq j + 2c, \text{ and} \tag{2}$$

$$|p - b_{i-1}| \leq |b_{i-2} - b_i| \text{ for } 3 \leq i \leq j + 2c. \tag{3}$$

Equation 2 is true, because otherwise $\{p, b_k\}$ would block $\{q', b_{i-1}\}$ instead of $\{p, b_i\}$.

Equation 3 is explained as follows: due to convexity and planarity, $\{p, b_{i+1}\} \in E$ implies $\{b_i, b_{i+2}\} \notin E$. This means the greedy algorithm added $\{p, b_{i+1}\}$ and discarded $\{b_i, b_{i+2}\}$ in later iteration (see Figure 11b).

Let $\gamma$ be an arbitrary constant. We call an edge *long* if its length is larger than $|q' - p| \cdot \gamma$. Because of Equation 2, once an edge $\{p, b_i\}$ is long, every edge $\{p, b_k\}$ is also long, for $1 \leq i < k \leq j + 2c$. Let $b_j$ be the point such that $\{p, b_j\}$ is the *last short* blocker, i.e.

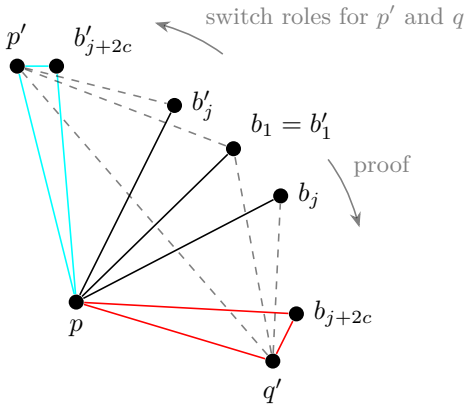
$$|p - b_j| \leq |q' - p| \cdot \gamma \text{ and} \tag{4}$$

$$|p - b_k| > |q' - p| \cdot \gamma \text{ for all } j + 1 \leq k \leq j + 2c. \tag{5}$$

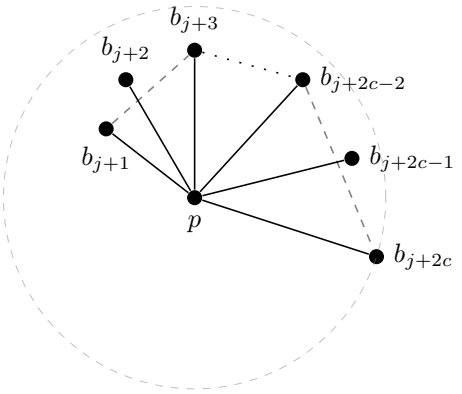
By this, the length of the last blocker $\{p, b_{j+2c}\}$ depends on $\gamma$ and $c$:

$$\begin{aligned} |p - b_{j+2c}| &\overset{\text{eq. 1}}{\leq} |q' - b_{j+2c-1}| \overset{\substack{\text{triangle}\\\text{inequality}}}{\leq} |q' - p| + |p - b_{j+2c-1}| \\ &\overset{\text{eq. 1}}{\leq} |q' - p| + |q' - b_{j+2c-2}| \overset{\substack{\text{triangle}\\\text{inequality}}}{\leq} \ldots \leq 2c \cdot |q' - p| + |p - b_j| \\ &\overset{\text{eq. 4}}{\leq} |q' - p| \cdot (2c + \gamma). \end{aligned} \tag{6}$$

Due to convexity, the points $b_{j+1}, \ldots, b_{j+2c}$ must be contained in the circle with centre $p$ and radius $|p - b_{j+2c}|$ (see Figure 11b). Therefore, their pairwise distances are bounded by its circumference. Upper and lower bounding the sum of the pairwise distances of tuples $b_{j+i}, b_{j+i+2}$ for $1 \leq i \leq 2c - 2$, it follows $c$ is bounded by a function dependent on $\gamma$:




**(a)** Regardless of the orientation, $|C_G(p,p')| \leq |\Delta_{pp'b'_{j+2c}}| + |\Delta_{pq'b_{j+2c}}|$.




**(b)** The points $b_{j+1}, \ldots, b_{j+2c}$ lie in the circle with centre $p$ and radius $|p - b_{j+2c}|$.

**Figure 11** Visualisation of the central proof steps of case i) in Theorem 16.

$$2c \cdot \gamma \cdot |q' - p| \overset{\text{eq. 5}}{<} \sum_{i=2}^{2c-1} |p - b_{j+i}| \overset{\text{eq. 3}}{\leq} \sum_{i=1}^{2c-2} |b_{j+i} - b_{j+i+2}| \overset{\text{eq. 6}}{\leq} 2\pi(2c+\gamma) \cdot |q' - p|$$
$$\iff \quad c < \pi \cdot \frac{\gamma - 1}{\gamma - 2\pi}.$$

For $\gamma = \mathcal{O}(1)$, it holds that $c = \mathcal{O}(1)$. Thus, the length of the last blocker $\{p, b_{j+2c}\}$ is bounded by a constant times $|p - q'|$.

This proof can be repeated with switched roles for $p'$ and $q$. Then, the points $b'_1, \ldots, b'_{j+2c}$ are ordered such that $\{p, b'_i\}$ blocks $\{p', b_{i-1}\}$ with $b_0 = q'$ (Figure 11a).

By definition of the point set $b_1, \ldots, b_{j+2c}$ (respectively $b'_1, \ldots, b'_{j+2c}$), it holds that $\{q', b_{j+2c}\} \in E$ and $\{p', b'_{j+2c}\} \in E$. Regardless of the orientation, the length of the shortest oriented cycle is bounded by $|C_G(p,p')| \leq |\Delta_{pq'b_{j+2c}}| + |\Delta_{pp'b'_{j+2c}}|$. This permits a constant dilation for the case $\{p, p'\} \in E$ as

$$\begin{aligned}
\operatorname{odil}(p,p') = \frac{|C_G(p,p')|}{|\Delta(p,p')|} &\leq \frac{|\Delta_{pq'b_{j+2c}}| + |\Delta_{pp'b'_{j+2c}}|}{|\Delta_{pp'q}|} \\
&\leq \max\left\{ \frac{|\Delta_{pq'b_{j+2c}}|}{|\Delta_{pp'q}|}, \frac{|\Delta_{pp'b'_{j+2c}}|}{|\Delta_{pp'q}|} \right\} \overset{\text{wlog}}{=} \frac{|p-q'| + |q'-b_{j+2c}| + |b_{j+2c}-p|}{|\Delta_{pp'q}|} \\
&\overset{\text{triangle inequality}}{\leq} \frac{2\cdot|p-q'| + 2\cdot|p-b_{j+2c}|}{|\Delta_{pp'q}|} \\
&\overset{\text{eq. 6}}{\leq} \frac{2\cdot|p-q'| + 2\cdot|p-q'|\cdot(2c+\gamma)}{|\Delta_{pp'q}|} = \frac{|p-q'|\cdot(4c+2\gamma+2)}{|p-p'|+|p'-q|+|q-p|} \\
&\overset{\alpha\text{-diamond-property}}{\leq} \frac{t_g\cdot|p-q|\cdot(4c+2\gamma+2)}{|\Delta_{pp'q}|} \leq t_g\cdot(4c+2\gamma+2) = \mathcal{O}(1).
\end{aligned}$$

For ii) $\{p, p'\} \notin E$, due to $\alpha$-diamond property, there is an undirected path $\Pi$ from $p$ to $p'$ in $T$ of length $|\Pi| \leq t_g \cdot |p - p'|$. Applying case i) gives factor $\lambda = \mathcal{O}(1)$ for each path edge $\{q', q''\} \in \Pi$. The dilation for the case $\{p, p'\} \notin E$ is bounded by

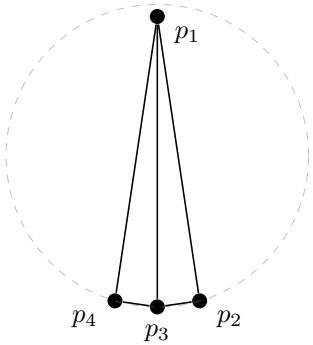



**Figure 12** Delaunay triangulation for convex point set. It is also a minimum dilation triangulation.

$$\text{odil}(p,p') = \frac{|C_G(p,p')|}{|\Delta(p,p')|} \leq \frac{\sum_{\{q',q''\}\in\Pi} |C_G(q',q'')|}{|\Delta(p,p')|}$$
$$\overset{\text{case i)}}{\leq} \frac{\lambda\cdot|\Pi|}{|\Delta(p,p')|} \leq \frac{\lambda\cdot t_g\cdot|p-p'|}{|\Delta(p,p')|} = \lambda\cdot t_g = \mathcal{O}(1). \qquad \blacktriangleleft$$

## 3.3 Other Triangulations

As the greedy triangulation leads to a plane oriented $\mathcal{O}(1)$-spanner for convex point sets, the question arises whether the $\alpha$-diamond property implies the existence of oriented spanners.

We already have seen that this is not the case for the greedy triangulation on general point sets. Likewise, the minimum weight triangulation has the $\alpha$-diamond property [17] but does not yield an $\mathcal{O}(1)$-spanner in general. This can be seen by the same example as for the greedy triangulation in Figure 10. By essentially the same argument, the minimum weight triangulation will have the same edges incident to $p_1$, which results in an oriented dilation of $\Omega(n)$. Whether the minimum weight triangulation is an oriented $\mathcal{O}(1)$-spanner for convex point sets, we leave as an open problem.

For the Delaunay triangulation the situation is even worse. The Delaunay triangulation has the $\alpha$-diamond property [16] and is the basis for many undirected plane spanner constructions [5]. However, for $n \geq 4$ its oriented dilation can be arbitrary large, no matter how we orient its edges, even for convex point sets.

Figure 12 shows the Delaunay triangulation $T = (P, E)$ of a convex point set of size 4. The shortest cycle $C_G(p_2, p_4)$ contains $p_1$, for any orientation of $T$. We can place $p_2, p_3, p_4$ arbitrarily close to each other without decreasing the radius of the circle through them. By placing $p_1$ in the circle and sufficiently far away from $p_2, p_3$ and $p_4$, the oriented dilation $\text{odil}(p_2, p_4) \geq \frac{|C_G(p_2,p_4)|}{|\Delta_{p_2p_3p_4}|}$ can be made arbitrary large. The example can be modified to include more points without changing $\text{odil}(p_2, p_4)$.

Finally, we consider orienting the undirected minimum dilation triangulation. However, essentially the same example as for the Delaunay triangulation shows that there are convex point sets for which any orientation has arbitrarily large oriented dilation. In Figure 12, consider the case in which we have fixed all positions except for the $y$-coordinates of $p_2$ and $p_4$. Let $y_0$ be the $y$-coordinate of $p_3$, and let $y_0 + \varepsilon$ be the $y$-coordinate of $p_2$ and $p_4$. By decreasing $\varepsilon > 0$, we can make sure that the minimum dilation triangulation chooses the same diagonal as the Delaunay triangulation, since the undirected dilation between $p_2$ and $p_4$ by the path through $p_3$ becomes arbitrarily close to 1. However, as in the Delaunay triangulation, the oriented dilation between $p_2$ and $p_4$ can be made arbitrarily large.

Thus, the quest for a triangulation of small oriented dilation for non-convex point sets remains open.

## 4 Conclusion and Outlook

Motivated by applications of geometric spanners, we introduced the concept of oriented geometric spanners. We provided a wide range of extremal and algorithmic results for oriented spanners in one and two dimensions.

Intriguingly, orienting the greedy triangulation yields a plane $\mathcal{O}(1)$-spanner for point sets in convex position, but not for general point sets. Furthermore, other natural triangulations like the Delaunay and the minimum weight triangulation do not lead to plane constant dilation spanners.

This raises the question of whether a plane constant dilation spanner exists and can be computed efficiently. If this is not possible, what is the lowest dilation that we can guarantee? Until now, even showing whether the greedy triangulation yields an $\mathcal{O}(n)$-spanner remains open.

As the concept of oriented spanners is newly introduced, it opens up many new avenues of research, for example:

- We know that the minimum one-page plane oriented spanner achieves a dilation of at most 5 (since this is the bound for the greedy algorithms) and that there are point sets where it has a dilation of 2. What is its worst-case dilation $2 \leq t \leq 5$? Can we compute it faster?
- We constructed a two-page plane oriented 2-spanner for any one-dimensional point set. Is 2 a tight upper bound on the dilation of a minimum two-page plane spanner? Is there an efficient algorithm to compute such a spanner?
- For two-dimensional point sets, the question of bounding minimum oriented dilation already arises without restricting to plane graphs. What is the worst-case dilation $2\sqrt{3} - 2 \leq t \leq 2$ of the minimum dilation oriented complete graph?
- Given an undirected geometric graph, can we efficiently compute an orientation minimising the dilation? For which graph classes is this possible?
- While undirected dilation compares the shortest path from $p$ to $p'$ in $G$ with the edge between them in the complete graph $K_n$, oriented dilation compares the shortest oriented cycle through $p$ to $p'$ in $G$ to the shortest triangle in $K_n$. An analogous measure in undirected graphs, *cyclic dilation*, would compare the shortest simple cycle in $G$ to the shortest triangle in $K_n$. Can we efficiently compute sparse graphs, in particular triangulations, of low cyclic dilation? Another measure of interest would be *detour dilation*, which compares the shortest path not using the edge $\{p, p'\}$ in $G$ (i.e. the shortest path in $G - \{p, p'\}$) with the triangle in $K_n$. Low detour dilation is a necessary condition for low oriented dilation, and actually at the core of our analysis of the greedy triangulation. Is there a triangulation with constant detour dilation? Can it be oriented to obtain constant oriented dilation?
- In the applications mentioned, it is desirable to reduce the number of bi-directional edges but it may not be necessary to avoid them completely. This opens up a whole new set of questions on the trade-off between directed dilation and the number of bi-directional edges. For instance, given a parameter $t$, compute the directed $t$-spanner with as few bi-directional edges as possible (and no bound on the number of oriented edges). Or, given a plane graph with certain edges marked as *one-way*, compute the orientation of these edges that minimises the directed dilation (while all other edges are bi-directional). For which families of graphs can this be done efficiently? For which is this problem NP-hard?

## References


1 Hugo A. Akitaya, Ahmad Biniaz, and Prosenjit Bose. On the spanning and routing ratios of the directed $\Theta_6$-graph. *Comput. Geom.*, 105-106:101881, 2022. doi:10.1016/j.comgeo.2022.101881.

2 Boris Aronov, Kevin Buchin, Maike Buchin, Bart Jansen, Tom De Jong, Marc van Kreveld, Maarten Löffler, Jun Luo, Bettina Speckmann, and Rodrigo Ignacio Silveira. Connect the dot: Computing feed-links for network extension. *J. Spatial Information Science*, 3:3–31, 2011. doi:10.5311/JOSIS.2011.3.47.

3 Michael A. Bekos, Martin Gronemann, and Chrysanthi N. Raftopoulou. Two-page book embeddings of 4-planar graphs. *Algorithmica*, 75(1):158–185, 2016. doi:10.1007/s00453-015-0016-8.

4 Prosenjit Bose, Aaron Lee, and Michiel H. M. Smid. On generalized diamond spanners. In *Proc. 10th Internat. Sympos. Algorithms and Data Struct.*, volume 4619 of *Lecture Notes in Computer Science*, pages 325–336. Springer-Verlag, 2007. doi:10.1007/978-3-540-73951-7_29.

5 Prosenjit Bose and Michiel Smid. On plane geometric spanners: A survey and open problems. *Comput. Geom. Theory Appl.*, 46(7):818–830, 2013. doi:10.1016/j.comgeo.2013.04.002.

6 Aléx F. Brandt, Miguel M. Gaiowski, Cid C. de Souza, and Pedro J. de Rezende. Minimum dilation triangulation: Reaching optimality efficiently. In *Proc. 26th Canad. Conf. Comput. Geom. (CCCG)*, 2014.

7 Martin Burkhart, Pascal Von Rickenbach, Rogert Wattenhofer, and Aaron Zollinger. Does topology control reduce interference? In *Proc. 5th ACM Internat. Sympos. Mobile Ad Hoc Networking and Computing*, pages 9–19, 2004. doi:10.1145/989459.989462.

8 Fan R. K. Chung, Frank Thomson Leighton, and Arnold L. Rosenberg. Embedding graphs in books: A layout problem with applications to vlsi design. *SIAM J. Algebraic Discrete Methods*, 8(1):33–58, 1987. doi:10.1137/0608002.

9 Lenore J. Cowen and Christopher G. Wagner. Compact roundtrip routing for digraphs. In *Proc. 10th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 885–886. SIAM, 1999. doi:10.5555/314500.315068.

10 Lenore J. Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *J. Algorithms*, 50(1):79–95, 2004. doi:10.1016/j.jalgor.2003.08.001.

11 Gautam Das and Deborah Joseph. Which triangulations approximate the complete graph? In *Optimal Algorithms*, volume 401 of *Lecture Notes in Computer Science*, pages 168–192. Springer-Verlag, 1989. doi:10.1007/3-540-51859-2_15.

12 Andrew Dobson and Kostas E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *Internat. J. Robotics Research*, 33(1):18–47, 2014. doi:10.1177/0278364913498.

13 Vida Dujmovic and David R. Wood. On linear layouts of graphs. *Discret. Math. Theor. Comput. Sci.*, 6(2):339–358, 2004. doi:10.46298/dmtcs.317.

14 David Eppstein. Spanning trees and spanners. In *Handbook of Computational Geometry*, pages 425–461. Elsevier, 2000. doi:h10.1016/B978-044482537-7/50010-3.

15 Panos Giannopoulos, Rolf Klein, Christian Knauer, Martin Kutz, and Dániel Marx. Computing geometric minimum-dilation graphs is NP-hard. *Internat. J. Comput. Geom. Appl.*, 20(2):147–173, 2010. doi:10.1142/S0218195910003244.

16 Robert L. (Scot) Drysdale III, Scott A. McElfresh, and Jack Snoeyink. On exclusion regions for optimal triangulations. *Discret. Appl. Math.*, 109(1-2):49–65, 2001. doi:10.1016/S0166-218X(00)00236-5.

17 J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete Comput. Geom.*, 7:13–28, 1992. doi:10.1007/BF02187821.

18 Christos Levcopoulos and Drago Krznaric. The greedy triangulation can be computed from the Delaunay triangulation in linear time. *Comput. Geom. Theory Appl.*, 14(4):197–220, 1999. doi:10.1016/S0925-7721(99)00037-1.

19 Christos Levcopoulos and Andrzej Lingas. Fast algorithms for greedy triangulation. *BIT*, 32(2):280–296, 1992. doi:10.1007/BF01994882.

20 Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. doi:10.1017/CBO9780511546884.

21 Christian Schindelhauer, Klaus Volbert, and Martin Ziegler. Geometric spanners with applications in wireless networks. *Comput. Geom. Theory Appl.*, 36(3):197–214, 2007. doi:10.1016/j.comgeo.2006.02.001.

22 Mihalis Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, 1989. doi:10.1016/0022-0000(89)90032-9.

# Online Coalition Formation Under Random Arrival or Coalition Dissolution

**Martin Bullinger**
Technical University of Munich, Germany

**René Romen**
Technical University of Munich, Germany

## Abstract

Coalition formation considers the question of how to partition a set of $n$ agents into disjoint coalitions according to their preferences. We consider a cardinal utility model with additively separable aggregation of preferences and study the online variant of coalition formation, where the agents arrive in sequence and whenever an agent arrives, they have to be assigned to a coalition immediately. The goal is to maximize social welfare. In a purely deterministic model, the greedy algorithm, where an agent is assigned to the coalition with the largest gain, is known to achieve an optimal competitive ratio, which heavily relies on the range of utilities.

We complement this result by considering two related models. First, we study a model where agents arrive in a random order. We find that the competitive ratio of the greedy algorithm is $\Theta\left(\frac{1}{n^2}\right)$, whereas an alternative algorithm, which is based on alternating between waiting and greedy phases, can achieve a competitive ratio of $\Theta\left(\frac{1}{n}\right)$. Second, we relax the irrevocability of decisions by allowing to dissolve coalitions into singleton coalitions, presenting a matching-based algorithm that once again achieves a competitive ratio of $\Theta\left(\frac{1}{n}\right)$. Hence, compared to the base model, we present two ways to achieve a competitive ratio that precisely gets rid of utility dependencies. Our results also give novel insights in weighted online matching.

**2012 ACM Subject Classification** Theory of computation → Online algorithms; Theory of computation → Algorithmic game theory

**Keywords and phrases** Online Algorithms, Coalition Formation, Online Matching

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.27

**Related Version** *Full Version*: https://arxiv.org/abs/2306.16965

**Funding** Deutsche Forschungsgemeinschaft, grants BR 2312/11-2, BR 2312/12-1.

**Acknowledgements** We would like to thank Thorben Tröbst and Viktoriia Lapshyna for valuable discussions.

## 1 Introduction

Coalition formation is a vibrant topic in multi-agent systems. The goal is to partition a set of $n$ agents into disjoint coalitions. We consider the framework of hedonic games, where the agents have preferences for the coalitions they are part of by disregarding externalities [18]. More specifically, we assume that each agent has cardinal utilities for each other agent and that utilities for coalitions are aggregated in an additively separable way by taking sums [7].

Most of the hedonic games literature considers an offline setting, where a fully specified instance is given. By contrast, in many real-life situations, such as the formation of teams in a company, the agents are not all present in the beginning but rather join an ongoing coalition formation process over time. With this motivation in mind, Flammini et al. [22] proposed an online version of hedonic games, where agents arrive one by one. Upon arrival, an agent reveals their preferences for coalitions containing the agents present thus far, and has to be added immediately and irrevocably to an existing coalition (or to a new singleton coalition).


Flammini et al. [22] seek to find algorithms that have high social welfare in a worst-case analysis against an adaptive adversary that is allowed to select an arbitrary instance and an arbitrary arrival order of the agents. The performance of an algorithm is measured with respect to its competitive ratio, that is, the worst-case ratio of the social welfare of the computed solution compared to the social welfare of the optimal offline solution. Their main result is that the greedy algorithm, which adds every agent to the best possible coalition upon arrival, has the optimal competitive ratio of $\Theta\left(\frac{1}{n}\frac{U_{min}}{U_{max}}\right)$, where $U_{max}$ and $U_{min}$ are the maximum and minimum absolute value of a non-zero single-agent utility in the adversarial instance, respectively.[1] In other words, for a fixed number of agents the competitive ratio is infinite if the maximum or minimum utility is not bounded.

Arguably, unbounded utilities give a lot of power to an adversary. The following example by Flammini et al. [22] exploits this. Consider the situation where the first two arriving agents have a mutual utility of 1. If an algorithm adds the second agent to the coalition of the first agent, then the adaptive adversary can set the utility of 1 as the minimum utility and add agents with large positive and negative utilities that bring no gain to this coalition. Hence, the welfare of the large positive utilities is lost, and the algorithm performs badly. Otherwise, if an algorithm puts the second agent into a singleton coalition, then the adversary can set the utility of other arriving agents to 0, and the loss of the welfare of the single positive edge leads to an unbounded competitive ratio.

As a consequence, we want to see whether we can perform better if the adversary has less power in two related models. First, we consider a model where the adversary is still capable of fixing a bad instance. However, agents arrive in a random order according to a permutation of the agents selected uniformly at random. We show that the greedy algorithm achieves a competitive ratio of $\Theta\left(\frac{1}{n^2}\right)$. Moreover, we present an alternative algorithm with a competitive ratio of $\Theta\left(\frac{1}{n}\right)$, which is based on alternating between waiting and greedy phases.

Second, we allow an algorithm to dissolve a coalition into singleton coalitions to revert bad previous decisions. We present a $\frac{1}{6}$-competitive online matching algorithm, which can be used to achieve a competitive ratio of $\Theta\left(\frac{1}{n}\right)$ for online coalition formation. Hence, compared to the deterministic model, we present a novel algorithm, whose competitive ratio gets precisely rid of utility dependencies. We conclude by showing boundaries for optimal algorithms in both settings. Moreover, we discuss why the worst-case analysis for our presented algorithm does not generalize for any algorithm.

## 2 Related Work

Hedonic games originated from economic theory [18] more than four decades ago. However, their broad consideration only started about two decades later [3, 7, 16]. In particular, Bogomolnaia and Jackson [7] introduced additively separable hedonic games (ASHGs), which are since then an ongoing subject of study. A large part of the research on ASHGs focuses on the computational complexity of stability measures [2, 9, 13, 17, 22, 23, 29, 30], but some more recent studies also consider economic efficiency in the sense of Pareto optimality [12, 19], popularity [8], or strategyproofness [21]. In general, maximizing social welfare is NP-hard, even if utilities are symmetric and only attain the values $-1$ and 1 [2]. Moreover, even though Bullinger [12] presents a polynomial-time algorithm to compute Pareto-optimal partitions for symmetric ASHG, the partitions computed by this algorithm can have negative social welfare. All of the literature discussed thus far considers ASHG in an offline setting.

[1] Flammini et al. [22] simplify the exposition by scaling the utilities such that $U_{min}$ is always 1.

The online variant only came under scrutiny very recently and has been researched far less. Flammini et al. [22] introduce the problem and focus on deterministic algorithms with guarantees to social welfare. Strictly speaking, they consider a variant of games equivalent to ASHGs, where utilities are scaled by a factor of 2. In particular, they consider games restricted by a maximum coalition size or maximum number of coalitions.

Moreover, there is a recent stream of research on dynamic models of coalition formation [4, 10, 15]. Instead of agents arriving over time, there exists an initial partition, which is altered over discrete time steps based on deviations of agents. Some of this literature explicitly deals with ASHGs or close variants [5, 6, 9, 14].

From the online algorithms literature, most related to our work is the online matching problem, which was first considered in the seminal paper by Karp et al. [27] considering online bipartite matching. In this work, an unweighted, bipartite graph is given, and the agents of one side appear online. The goal is to find a matching of maximum cardinality. Karp et al. [27] introduce the famous ranking algorithm, which achieves a competitive ratio of $1 - \frac{1}{e}$. An overview of this research is presented in the very recent book chapter by Huang and Tröbst [25].

Our model of online coalition formation can be viewed as a generalization of the setting by Karp et al. [27], with the following modifications: (i) there are edge weights, (ii) all vertices arrive online, (iii) the underlying graph is not necessarily bipartite, and (iv) coalitions can be arbitrary subsets of agents. While condition (iv) is specific to coalition formation, conditions (i)-(iii) have been studied in the literature, albeit, to the best of our knowledge, not in the combination of all three. Feldman et al. [20] consider condition (i), i.e., a bipartite setting with edge weights, where one side of the vertices arrives online. They show that the natural greedy algorithm is $\frac{1}{2}$-competitive, and they provide an algorithm matching the competitive ratio of $1 - \frac{1}{e}$ from Karp et al. [27]. An important condition in this setting is *free disposal*, which essentially requires that a previous matching can be dissolved upon the arrival of a better option. Wang and Wong [31] consider condition (ii), i.e., online bipartite matching where both sides arrive online. They can beat the greedy algorithm by a primal-dual algorithm achieving a competitive ratio of $0.532 < 1 - \frac{1}{e}$. Finally, Huang et al. [24] consider the conjunction of conditions (ii) and (iii), i.e., fully online (non-bipartite) matching. They extend the ranking algorithm to this setting and show that it is 0.5211-competitive. Both Wang and Wong [31] and Huang et al. [24] show that a competitive ratio of $1 - \frac{1}{e}$ is impossible to achieve in their respective settings. Interestingly, the competitive ratio of $1 - \frac{1}{e}$ can be beaten in the random arrival model [26, 28].

## 3 Preliminaries

In this section, we present our model. For an integer $i \in \mathbb{N}$, we define $[i] := \{1, \dots, i\}$. Additionally, for any set $N$, define $\binom{N}{2} := \{e \subseteq N \colon |e| = 2\}$.

### 3.1 Additively Separable Hedonic Games

Let $N$ be a finite set of $n$ *agents*. Any subset of $N$ is called a *coalition*. We denote the set of all possible coalitions containing agent $i \in N$ by $\mathcal{N}_i = \{C \subseteq N \colon i \in C\}$. A *coalition structure* (or *partition*) is a partition of the agents. Given an agent $i \in N$ and a partition $\pi$, let $\pi(i)$ denote the coalition of $i$, i.e., the unique coalition $C \in \pi$ with $i \in C$.

A *hedonic game* is a pair $(N, \succsim)$ consisting of a set $N$ of agents and a preference profile $\succsim = (\succsim_i)_{i \in N}$, where $\succsim_i$ is a weak order over $\mathcal{N}_i$ that represents the preferences of agent $i$. A hedonic game is called an *additively separable hedonic game* (ASHG) if there exists a

complete, undirected, and weighted graph $G = (N, E, w)$ with edge set $E = \binom{N}{2}$ and weight function $w\colon E \to \mathbb{Q}$, such that, for every agent $i \in N$ and every pair of coalitions $C, C' \in \mathcal{N}_i$, it holds that $C \gtrsim_i C'$ if and only if $\sum_{j\in C} w(\{i,j\}) \geq \sum_{j\in C'} w(\{i,j\})$ [7].[2] We then speak of the ASHG *given by* $G$. We abbreviate $w(i,j) = w(\{i,j\})$. Moreover, since we only consider complete graphs, we shorten notation and write $G = (N, w)$, where $w\colon \binom{N}{2} \to \mathbb{Q}$, instead of $G = (N, E, w)$ to fully specify an underlying graph. For an agent $i \in N$ and a coalition $C \in \mathcal{N}_i$ or a partition $\pi$, we define the *utility* of $i$ for $C$ or $\pi$ by $u_i(C) := \sum_{j\in C} w(i,j)$ and $u_i(\pi) := u_i(\pi(i))$, respectively.

Additionally, we extend the weight function to sets of edges $F \subseteq \binom{N}{2}$ by $w(F) := \sum_{e\in F} w(e)$. A *matching* is a coalition structure $\pi$ such that, for all $C \in \pi$, it holds that $|C| \leq 2$. A matching $\pi$ is represented by its edge set $M(\pi) := \{C \in \pi\colon |C| = 2\} \subseteq \binom{N}{2}$. We then write $w(\pi) := w(M(\pi))$ for the weight of matching $\pi$.

## 3.2 Online Coalition Formation

In this section, we introduce our model of online coalition formation and appropriate objectives. Consider an ASHG given by $G = (N, w)$. Given a subset of agents $N' \subseteq N$, let $G[N']$ denote the subgraph induced by agent set $N'$. Moreover, given a partition $\pi$ of $N$ and a subset of agents $N' \subseteq N$, we define $\pi[N']$ as the *partition restricted to* $N'$ as $\pi[N'] := \{C \cap N'\colon C \in \pi, C \cap N' \neq \emptyset\}$. Specifically, if $N' = N \setminus \{i\}$ for some agent $i \in N$, we write $\pi - i$ instead of $\pi[N']$.

In an online setting, previous decisions influence the capabilities of an algorithm to form a partition in the next step. Given a partition $\pi$ and an agent $i$ not covered by $\pi$, let $\mathcal{A}(\pi, i)$ denote the set of *available partitions*, when the tentative partition is $\pi$ and the newly arriving agent is $i$. As a default, we assume the standard setting where $\mathcal{A}(\pi, i) = \mathcal{A}^S(\pi, i) := \{\pi'\colon \pi' - i = \pi\}$. We also consider algorithms that have the capability to dissolve a coalition completely. We say that an algorithm acts under *free dissolution* if $\mathcal{A}(\pi, i) = \mathcal{A}^D(\pi, i) := \mathcal{A}^S(\pi, i) \cup \bigcup_{C\in\pi, j\in C}\{(\pi \setminus \{C\}) \cup \{\{i,j\}\} \cup \{\{k\}\colon k \in C \setminus \{j\}\}\}$. Free dissolution is the natural extension of free disposal by Feldman et al. [20] in the domain of matching adapted to coalitions of size larger than 2. In addition, we define $\Sigma(N) := \{\sigma\colon [|N|] \to N \text{ bijective}\}$ as the set of all *orders* of the agent set $N$.

An instance of an *online coalition formation problem* is a tuple $(G, \sigma)$, where $G = (N, w)$ defines an ASHG and $\sigma \in \Sigma(N)$. An *online coalition formation algorithm* for instance $(G, \sigma)$ gets as input the sequence $G_1, \ldots, G_n$, where, for every $i \in [n]$, $G_i = G[\{\sigma(1), \ldots, \sigma(i)\}]$. Then, for every $i \in [n]$, the algorithm produces a partition $\pi_i$ of $\{\sigma(1), \ldots, \sigma(i)\}$ such that the algorithm has only access to $G_i$ and for $i \geq 2$, it holds that $\pi_i \in \mathcal{A}(\pi_{i-1}, \sigma(i))$.

The output of the algorithm is the partition $\pi_n$. Given an online coalition formation algorithm $ALG$, let $ALG(G, \sigma)$ be its output for instance $(G, \sigma)$.

In other words, the algorithm iteratively builds a partition such that, whenever an agent arrives, the only knowledge is the game restricted to the present agents, and the algorithm has to irrevocably assign the new agent to an existing coalition or start a new coalition (or, under free dissolution, dissolve any coalition before making its decision). If an online coalition formation algorithm creates a matching in any step, we speak of an *online matching algorithm*.

Our benchmark algorithm is the greedy algorithm as introduced by Flammini et al. [22].

[2] Since our focus will be on social welfare, the consideration of *undirected* graphs is without loss of generality because the social welfare of a partition is invariant under the symmetrization $w^S(\{x,y\}) = \frac{1}{2}(w_x(y) + w_y(x))$ given directed edges with weights $w_i(j)$.

▶ **Definition 1** (Greedy algorithm). *On input* $(G, \sigma)$*, in the* $i$*th step,* $i \geq 2$*, the* greedy algorithm *(GDY) forms* $\pi_i = \arg\max_{\pi \in \mathcal{A}(\pi_{i-1}, \sigma(i))} \mathcal{SW}(\pi)$ *if there exists* $\pi \in \mathcal{A}(\pi_{i-1}, \sigma(i))$ *with* $\mathcal{SW}(\pi) > \mathcal{SW}(\pi_{i-1})$*, and* $\pi_i = \pi_{i-1} \cup \{\{\sigma(i)\}\}$*, otherwise.*

Hence, *GDY* assigns each arriving agent to the available coalition such that the increase in social welfare is maximized, or creates a new singleton coalition if no increase is possible.

## 3.3 Competitive Analysis

The competitive analysis needs a quantifiable objective and we follow Flammini et al. [22] by considering social welfare. The *social welfare* of a partition $\pi$ is defined as $\mathcal{SW}(\pi) = \sum_{i \in N} u_i(\pi)$. A partition $\pi$ is said to be *welfare-optimal* if, for every partition $\pi'$, it holds that $\mathcal{SW}(\pi) \geq \mathcal{SW}(\pi')$. Given a hedonic game $G$, let $\pi^*(G)$ be a welfare-optimal partition. We say that an online coalition formation algorithm $ALG$ is *c-competitive*[3] if

$$\inf_{G} \min_{\sigma \in \Sigma(N)} \frac{\mathcal{SW}(ALG(G, \sigma))}{\mathcal{SW}(\pi^*(G))} \geq c.$$

Equivalently, this means that, for all instances $(G, \sigma)$, it holds that $\mathcal{SW}(ALG(G, \sigma)) \geq c\,\mathcal{SW}(\pi^*(G))$. The *competitive ratio* of an algorithm $ALG$, denoted by $c_{ALG}$, is the maximum $c$ such that $ALG$ is $c$-competitive. Note that the competitive ratio is always at most 1.

In addition, we consider online coalition formation with a random arrival order, where we assume that the arrival order is selected uniformly at random. In the random arrival model, the competitive ratio of an algorithm $ALG$ is defined as

$$\inf_{G} \frac{\mathbb{E}_\sigma\left[\mathcal{SW}(ALG(G, \sigma))\right]}{\mathcal{SW}(\pi^*(G))} \geq c.$$

There, the expectation is over the uniform selection of an arrival order $\sigma$ from $\Sigma(N)$. Note that if $\pi$ is a matching, then $w(\pi) = \frac{1}{2}\mathcal{SW}(\pi)$. Hence, in the competitive analysis of online matching algorithms, we can as well consider the weight of matchings instead of their social welfare.

# 4 Random Arrival Model

In this section, we analyze algorithms aiming at achieving a high social welfare in the random arrival model. Interestingly, while in the deterministic arrival model, the specific utility values affect the competitive ratio of the greedy algorithm (and any other algorithm) [22], in the random arrival model, the dependency is solely on the number of agents. For our analysis of algorithms in this section we use the notation $x \prec^\sigma y$ to say that $\sigma^{-1}(x) < \sigma^{-1}(y)$ for $x, y \in N$ and an arrival order $\sigma$.

▶ **Theorem 2.** *The competitive ratio of GDY for ASHGs under a random arrival order satisfies* $\Theta\left(\frac{1}{n^2}\right)$.

**Proof.** First, we show that the competitive ratio of *GDY* satisfies $O\left(\frac{1}{n^2}\right)$ by providing an example where it performs "bad". Let $\epsilon > 0$ and consider the ASHG given by $G^\epsilon = (N, w^\epsilon)$ depicted in Figure 1. The agent set is $N = X \cup Y \cup \{a, b\}$, where $|X| = |Y| = m$. Utilities

[3] Here, we use the convention that $\frac{0}{0} = 1$ and $\frac{x}{0} = 0$ for any $x \in \mathbb{Q}_{<0}$. Also, note that Flammini et al. [22] define the competitive ratio in the inverse way such that it is always at least 1. Here, we prefer the more common definition in the online matching literature.

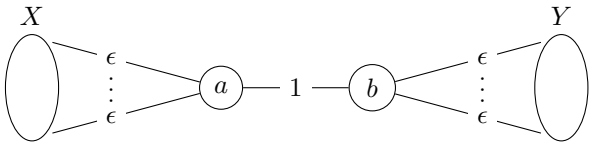



**Figure 1** The example contains $n = 2m + 2$ agents. There are $m$ agents in each of the sets $X$ and $Y$. The utility between $a$ and any agent in $X$ and $b$ and any agent in $Y$ is $\epsilon$. All omitted edges represent utilities of $-1$.

are given by $w^\epsilon(a,b) = 1$, $w^\epsilon(a,x) = w^\epsilon(b,y) = \epsilon$ for $x \in X$ and $y \in Y$, and all other weights are $-1$. Clearly, for sufficiently small $\epsilon$, the optimal solution has a value of 2 (with $\{a,b\}$ the only non-singleton coalition). By inspecting the limit case for $\epsilon$ tending to 0, the value of the greedy algorithm (and therefore also its competitive ratio) is at most twice the probability of forming $\{a,b\}$, i.e.,

$$\begin{aligned} c_{GDY} &= \inf_G \frac{\mathbb{E}_\sigma\left[\mathcal{SW}(GDY(G,\sigma))\right]}{\mathcal{SW}(\pi^*(G))} \leq \inf_{\epsilon>0} \frac{\mathbb{E}_\sigma\left[\mathcal{SW}(GDY(G^\epsilon,\sigma))\right]}{2} \\ &\leq \mathbb{P}_\sigma(\{a,b\} \in GDY(G^\epsilon,\sigma)). \end{aligned}$$

We compute

$$\begin{aligned} \mathbb{P}_\sigma(\{a,b\} \in GDY(G^\epsilon,\sigma)) &= \mathbb{P}_\sigma(\{a,b\} \in GDY(G^\epsilon,\sigma) \mid a \prec^\sigma b)\mathbb{P}_\sigma(a \prec^\sigma b) \\ &\quad + \mathbb{P}_\sigma(\{a,b\} \in GDY(G^\epsilon,\sigma) \mid b \prec^\sigma a)\mathbb{P}_\sigma(b \prec^\sigma a) \\ &= 2\mathbb{P}_\sigma(\{a,b\} \in GDY(G^\epsilon,\sigma) \mid a \prec^\sigma b)\mathbb{P}_\sigma(a \prec^\sigma b). \end{aligned}$$

The second equality follows by symmetry. Next, we sum over all possible arrival positions of $b$ by summing over the number of alternatives that arrive before $b$ in addition to $a$. Note that if more than $m$ agents arrive before $b$, excluding $a$, then, by the pigeonhole principle, some $x \in X$ arrives before $b$ and forms a coalition with $a$. This prevents the coalition $\{a,b\}$ from forming so all terms of the sum for $i > m$ are 0. Conditioned on $a$ arriving before $b$, the coalition $\{a,b\}$ forms if and only if all $i$ agents arriving before $b$ are from $Y$ since those agents will not form a coalition with $a$. We derive[4]

$$\begin{aligned} c_{GDY} &\leq 2\sum_{i=0}^{m} \mathbb{P}_\sigma(\{a,b\} \in GDY(G^\epsilon,\sigma) \mid a \prec^\sigma b, \sigma^{-1}(b) = i+2) \\ &\qquad \cdot \mathbb{P}_\sigma(a \prec^\sigma b \mid \sigma^{-1}(b) = i+2)\mathbb{P}_\sigma(\sigma^{-1}(b) = i+2) \\ &= 2\sum_{i=0}^{m} \mathbb{P}_\sigma(\{d\colon d \prec^\sigma b\} \setminus \{a\} \subseteq Y \mid a \prec^\sigma b, \sigma^{-1}(b) = i+2) \\ &\qquad \cdot \mathbb{P}_\sigma(a \prec^\sigma b \mid \sigma^{-1}(b) = i+2)\mathbb{P}_\sigma(\sigma^{-1}(b) = i+2). \end{aligned}$$

We compute all individual terms in the previous sum. First, the probability that the $i$ agents arriving before $b$ are all from $Y$ is $\mathbb{P}_\sigma(\{d\colon d \prec^\sigma b\} \setminus \{a\} \subseteq Y \mid a \prec^\sigma b, \sigma^{-1}(b) = i+2) = \frac{\binom{m}{i}}{\binom{2m}{i}}$, i.e., the number of possibilities to draw $i$ agents from $Y$ divided by the number of possibilities to draw $i$ agents from $Y \cup X$.

[4] For the first inequality, note that it holds that $\mathbb{P}(A \mid B)\mathbb{P}(B) = \mathbb{P}(A,B) = \sum_C \mathbb{P}(A,B,C) = \sum_C \mathbb{P}(A \mid B,C)\mathbb{P}(B,C) = \sum_C \mathbb{P}(A \mid B,C)\mathbb{P}(B \mid C)\mathbb{P}(C)$ for arbitrary events $A$, $B$, and $C$ and probability measures $\mathbb{P}$.

Second, the probability that $a$ arrives before $b$ when $b$ arrives in position $i+2$ is $\mathbb{P}_\sigma(a \prec^\sigma b \mid \sigma^{-1}(b) = i+2) = \frac{i+1}{2m+1}$ because we have $i+1$ chances to draw $a$ among the remaining $2m+1$ alternatives. Finally, due to the random arrival order, the probability that agent $b$ arrives in a certain fixed position is $\mathbb{P}_\sigma(\sigma^{-1}(b) = i+2) = \frac{1}{2m+2}$. Together, we obtain

$$c_{GDY} \leq 2\sum_{i=0}^{m} \frac{\binom{m}{i}}{\binom{2m}{i}} \frac{i+1}{2m+1}\frac{1}{2m+2} = \frac{2}{m^2+3m+2} \in O\left(\frac{1}{m^2}\right) = O\left(\frac{1}{n^2}\right).$$

Next, we show that the competitive ratio of $GDY$ satisfies $\Omega\left(\frac{1}{n^2}\right)$. Consider an arbitrary ASHG given by $G = (N, w)$ and let $E^+(G) = \left\{e \in \binom{N}{2} \colon w(e) > 0\right\}$ be the set of agent pairs with positive weights. Then, the welfare-optimal partition $\pi^*(G)$ satisfies $\mathcal{SW}(\pi^*(G)) \leq 2\sum_{e\in E^+(G)} w(e)$. Furthermore, the social welfare of $GDY$ is at least twice the utility between the first arriving pair of agents from $E^+(G)$. Indeed, until the arrival of a pair of agents with positive utility, every agent is assigned to a singleton coalition. Thus, since every pair in $E^+(G)$ has equal probability to be the first such pair, the expected social welfare of $GDY$ is at least the average, i.e., $\mathbb{E}_\sigma[\mathcal{SW}(GDY(G,\sigma))] \geq \frac{\sum_{e\in E^+(G)} 2w(e)}{|E^+(G)|}$ and the competitive ratio is then at least

$$c_{GDY} = \inf_G \frac{\mathbb{E}_\sigma[\mathcal{SW}(GDY(G,\sigma))]}{\mathcal{SW}(\pi^*(G))} \geq \inf_G \frac{\frac{\sum_{e\in E^+(G)} 2w(e)}{|E^+(G)|}}{2\sum_{e\in E^+(G)} w(e)} = \inf_G \frac{1}{|E^+(G)|} \in \Omega\left(\frac{1}{n^2}\right).$$

Altogether, we have shown that $c_{GDY}$ is of order $\Theta\left(\frac{1}{n^2}\right)$. ◀

The natural question is whether we can obtain better algorithms than the greedy algorithm. A simple attempt to achieve a better algorithm is to make use of randomization. The performance of the greedy algorithm was bounded because in the worst case, the value of greedy is equal to the average weight of a positive edge. However, we can easily achieve the average weight of a random matching, improving the performance to $1/n$. For simplicity, we assume first that $n$ is even and known to the algorithm in advance. We first analyze a simple online matching algorithm.

▶ **Definition 3** (Random matching algorithm). *The* random matching algorithm *(RMA) leaves the first $\frac{n}{2}$ agents unmatched. Then, we select a bijection $\phi\colon \left[\frac{n}{2}\right] \to \left[\frac{n}{2}\right]$ uniformly at random. For $1 \leq i \leq \frac{n}{2}$, if the weight between the $\left(\frac{n}{2}+i\right)$th agent and the $\phi(i)$th agent is positive, then these are matched. Otherwise, the $\left(\frac{n}{2}+i\right)$th remains unmatched.*

We determine the competitive ratio of *RMA*.

▶ **Proposition 4.** *RMA has a competitive ratio of $\Theta\left(\frac{1}{n}\right)$ for matching instances under a random arrival order when $n$ is known.*

**Proof.** Consider an instance given by $G = (N, w)$ such that there exists agents $a, b \in N$ with $w(a, b) = 1$, and $w(x, y) = 0$ if $\{x, y\} \neq \{a, b\}$. Then, the maximum weight matching has weight 1, but *RMA* has only a chance of

$$\frac{\binom{2}{1}\binom{n-2}{\frac{n}{2}-1}}{\binom{n}{\frac{n}{2}}} \cdot \frac{1}{n} = \frac{n}{2(n-1)} \cdot \frac{1}{n} = \Theta\left(\frac{1}{n}\right),$$

to have $a$ and $b$ in the same coalition. There, the first part of the product is the chance that $a$ and $b$ appear in different stages of the algorithm, and the factor of $1/n$ is the probability that they are matched conditioned on them arriving in different stages. Hence, the competitive ratio of *RMA* satisfies $O\left(\frac{1}{n}\right)$.

On the other hand, in an execution of *RMA*, every positive edge has a probability of $\frac{n}{2(n-1)} \cdot \frac{1}{n}$ to contribute to the computed matching. Let $E^+ = \left\{e \in \binom{N}{2} \colon w(e) > 0\right\}$. Hence, $\mathbb{E}_\sigma[RMA(G,\sigma)] = \sum_{e \in E^+} \frac{n}{2(n-1)} \cdot \frac{1}{n} w(e) \geq \frac{1}{2n} w(E^+)$. Since any matching is of weight at most $w(E^+)$, we conclude that

$$c_{RMA} \geq \frac{1}{2n} = \Omega\left(\frac{1}{n}\right). \qquad \blacktriangleleft$$

Our next goal is to derandomize this algorithm while maintaining the same competitive ratio. A natural way to do this is to have the first half of the agents form singleton coalitions, and the second half of the agents join the best available coalition. This yields a deterministic algorithm achieving a competitive ratio of $\frac{1}{n}$. First, we maintain the assumption that $n$ is even and known to the algorithm beforehand. However, after we have analyzed this algorithm, we will show that we can drop this additional assumption and that a variation of the algorithm still achieves the same competitive ratio asymptotically.

▶ **Definition 5** (Waiting greedy algorithm)**.** *The* waiting greedy algorithm *(WGDY) places the first $\frac{n}{2}$ agents in singleton coalitions. Then, for the remaining $\frac{n}{2}$ agents, it assigns coalitions greedily.*

The upper bound of the performance of *WGDY* is again attained by the game depicted in Figure 1. The analysis is more involved and relies on investigating the distribution of the agents in the second phase. We defer the complete proof to the full version and restrict attention to the lower bound. The full version also contains all other missing proofs.

▶ **Theorem 6.** *The competitive ratio of WGDY for ASHGs under a random arrival and known and even $n$ is $\Theta\left(\frac{1}{n}\right)$.*

**Proof of lower bound.** We show that $WGDY \in \Omega\left(\frac{1}{n}\right)$. Consider an arbitrary ASHG given by $G = (N, w)$. Let $\Pi(n) = \left\{(A,B) \colon A \cup B = N, |A| = |B| = \frac{n}{2}\right\}$ be the set of partitions of the agent set $N$ into two equally-sized subsets.

Let $(A,B) \in \Pi(n)$. Define $E^+(A,B) = \left\{e = \{a,b\} \in \binom{N}{2} \colon w(e) > 0, a \in A, b \in B\right\}$. We claim that if the agents in $A$ and $B$ arrive in the first and second stage, respectively, then the weight of the obtained partition is at least $\frac{1}{n} w(E^+(A,B))$. Let $P_{A,B}$ denote the event that the partition $(A,B)$ realizes.

Consider an arbitrary agent $b \in B$. Let $a_1(b), \ldots, a_{r(b)}(b) \in A$ be the agents in $A$ such that $\{b, a_i(b)\} \in E^+(A,B)$, where $r(b) \in \mathbb{N}$ is their number. Assume that $w(a_1(b), b) \geq w(a_2(b), b) \geq \cdots \geq w(a_{r(b)}(b), b)$. Let $i \in [r(b)]$. In the event $P_{A,B}$, an agent $b \in B$ arrives as the $(\frac{n}{2} + i)$th agent (i.e., the $i$th agent within $B$) with a probability of $\frac{2}{n}$. Moreover, when $b$ arrives as the $(\frac{n}{2} + i)$th agent, then at most $i - 1$ coalitions have formed so far, and therefore some agent in $\{a_1(b), \ldots, a_i(b)\}$ is still in a singleton coalition. Hence, the increase in weight caused by $b$ is at least the weight of forming a singleton coalition with the worst partner in $\{a_1(b), \ldots, a_i(b)\}$, that is, $w(a_i(b), b)$. Let $I_b$ be the random variable denoting the gain in social welfare caused by the arrival of $b$.

Hence, in the event $P_{A,B}$, the attained expected social welfare satisfies

$$\begin{aligned}
\mathbb{E}_\sigma[\mathcal{SW}(\mathit{WGDY}(G,\sigma)) \mid P_{A,B}] &\geq \sum_{b\in B} \mathbb{E}_\sigma[I_b \mid P_{A,B}] \\
&= \sum_{b\in B}\sum_{i=1}^{r(b)} \mathbb{E}_\sigma\left[I_b \mid P_{A,B}, \sigma^{-1}(b) = \frac{n}{2}+i\right] \mathbb{P}_\sigma\left(\sigma^{-1}(b) = \frac{n}{2}+i \mid P_{A,B}\right) \\
&= \sum_{b\in B}\sum_{i=1}^{r(b)} \mathbb{E}_\sigma\left[I_b \mid P_{A,B}, \sigma^{-1}(b) = \frac{n}{2}+i\right] \frac{2}{n} \\
&\geq \sum_{b\in B}\sum_{i=1}^{r(b)} w(a_i(b),b)\frac{2}{n} = \frac{2}{n}\sum_{b\in B}\sum_{e\in E^+(A,B)\colon b\in e} w(e) = \frac{2}{n}w(E^+(A,B)).
\end{aligned}$$

Hence, the expected social welfare of the partition computed by *WGDY* is

$$\mathbb{E}_\sigma\left[\mathcal{SW}(\mathit{WGDY}(G,\sigma))\right] \geq \frac{1}{|\Pi(n)|}\sum_{(A,B)\in\Pi(n)} \frac{2}{n} w\left(E^+(A,B)\right) \geq \frac{1}{2}\frac{2}{n} w\left(E^+\right). \tag{1}$$

In the first inequality, we have used that each partition in $\Pi(n)$ is realized with equal probability, and in the second inequality, we have used that an edge in $E^+ = \left\{e \in \binom{N}{2}\colon w(e) > 0\right\}$ is in $E^+(A,B)$ for at least half of the partitions in $\Pi$. The latter holds because there are $n^2/4$ edges between $A$ and $B$ and $n(n-1)/2$ edges in total.

Additionally, the social welfare of any partition is at most $2w(E^+)$. Combining this with Equation (1), we obtain

$$c_{\mathit{WGDY}} \geq \frac{\frac{w(E^+)}{n}}{2w(E^+)} = \frac{1}{2n} \in \Omega\left(\frac{1}{n}\right). \qquad \blacktriangleleft$$

Next, we show that we can still use a variant of this algorithm even if we do not know $n$. The idea is to run *WGDY* on an exponentially growing number of agents repeatedly, possibly not finishing the last iteration.

▶ **Definition 7** (Iterated waiting algorithm). *The* iterated waiting algorithm *(IWA) alternates between placing $2^i$ agents in singleton coalitions and then assigning $2^i$ agents to coalitions with the previous $2^i$ agents greedily. The parameter is set to $i = 0$ at the start and is increased by 1 after $2^{i+1}$ agents arrive.*

The lower bound in the analysis follows, because *IWA* completes an execution of *WGDY* for a sufficiently large number of agents. For the upper bound, we can once again use the game in Figure 1. The analysis is even more involved as in Theorem 6 because we need to consider dependencies of the distributions of the agents between the phases of *IWA*.

▶ **Theorem 8.** *IWA has a competitive ratio of $c \in \Theta\left(\frac{1}{n}\right)$.*

## 5 Deterministic Model under Free Dissolution

In this section, we consider the model with deterministic arrivals under free dissolution. For bipartite online matching instances, where one side of the agents is present offline, Feldman et al. [20] show that *GDY*, i.e., transitioning to best coalitions within $\mathcal{A}^D(\pi,i)$ achieves a competitive ratio of $\frac{1}{2}$. However, if all vertices arrive online, then *GDY* does not even achieve a constant competitive ratio.

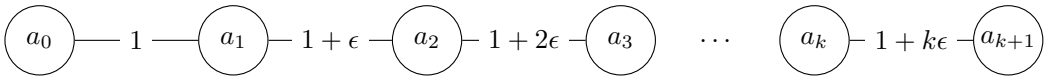



**Figure 2** Family of instances for upper bound of the competitive ratio of *GDY* for matching instances under free dissolution.

▶ **Proposition 9.** *In the deterministic model with free dissolution, GDY has a competitive ratio of* $\Theta(\frac{1}{n})$ *in the matching domain.*

**Proof.** First, note that *GDY* will definitely match a pair of agents with the highest possible weight $W_{\max}$. Moreover, every matching has weight at most $\frac{n}{2}W_{\max}$, and therefore $c_{GDY} \in \Omega(\frac{1}{n})$.

For the upper bound, consider the family of instances depicted in Figure 2. Let $\epsilon > 0$ and $k \in \mathbb{N}$ even. Consider $G^{k,\epsilon} = (N^{k,\epsilon}, w^{k,\epsilon})$ where $N^{k,\epsilon} = \{a_i \colon 0 \leq i \leq k+1$ and, for $i \in [k+1]$, $w^{k,\epsilon}(a_{i-1}, a_i) = 1 + (i-1)\epsilon$. All other weights are set to 0. Let the arrival order be $(a_0, a_1, \ldots, a_k, a_{k+1})$. Let $M^*(G^{k,\epsilon})$ be the maximum weight matching for the instance given by $G^{k,\epsilon}$.

Then, $M^*(G^{k,\epsilon}) = \left\{\{a_{2i}, a_{2i+1}\} \colon 0 \leq i \leq \frac{k}{2}\right\}$ and *GDY* outputs the matching $\{\{a_k, a_{k+1}\}\}$. Hence,

$$\inf_{\epsilon>0} \frac{\mathcal{SW}(GDY(G^{k,\epsilon}, \sigma^k))}{\mathcal{SW}(M^*(G^{k,\epsilon}, \sigma^k))} = \inf_{\epsilon>0} \frac{1+k\epsilon}{\frac{k}{2} + \frac{1}{4}k(k+2)\epsilon} = \frac{2}{k} \in \Theta\left(\frac{1}{n}\right). \qquad \blacktriangleleft$$

The non-zero edges in the construction of the previous proposition even induce *bipartite* graphs, and it is therefore essential for the result by Feldman et al. [20] that one side of the agents is present offline. Proposition 9 indicates that even achieving a constant competitive ratio is a non-trivial task for an online matching algorithm for weighted graphs. In fact, as we will see in Section 6, the usual result of achieving a competitive ratio of $1/2$ with some algorithm [27, 20, 31] is impossible. Still, we can modify the greedy algorithm to achieve a constant competitive ratio.

▶ **Definition 10** (Dissolution threshold algorithm)**.** *Given a matching* $\pi$ *and a newly arriving vertex* $i$*, the* dissolution threshold algorithm *(DTA) is the greedy algorithm for the available matchings* $\mathcal{A}(\pi, i) = \{(\pi - j) \cup \{\{i, j\}\} \colon \{j\} \in \pi\} \cup \{(\pi \setminus \{C\}) \cup \{\{i, j\}, \{k\}\} \colon C = \{j, k\} \in \pi, w(i, j) \geq 2w(j, k)\}$.

In words, *DTA* only dissolves a matched pair if the weight of the new edge compared to the weight of the dissolved edge is larger by at least a factor of 2.

▶ **Theorem 11.** *In the deterministic model with free dissolution, DTA has a competitive ratio of* $\frac{1}{6}$ *in the matching domain.*

**Proof.** We show first that $c_{DTA} \leq \frac{1}{6}$. To this end, we define a family of instances $(G^{k,\epsilon})_{k\geq 1, \epsilon > 0}$ with a sufficiently large gap between the social welfare of the algorithmic and optimal solutions. The construction is depicted in Figure 3.

Let $G^k = (N^k, w^k)$ where $N^k = \{a_i, b_i \colon 0 \leq i \leq k+1\}$. The edge weights are given by $w(a_i, b_i) = 2^{i+1} - \epsilon$ for $0 \leq i \leq k$, $w(a_{k+1}, b_{k+1}) = 2^{k+1} - \epsilon$, and $w(a_i, a_{i+1}) = 2^i$ for $0 \leq i \leq k$. All other weights are set to 0. The arrival order $\sigma^k$ is $(a_0, a_1, b_0, a_2, b_1, \ldots, a_{k+1}, b_k, b_{k+1})$.

Consider an execution of *DTA* for $(G^{k,\epsilon}, \sigma^k)$. It is easy to see that $DTA(G^{k,\epsilon}, \sigma^k) = \{\{a_k, a_{k+1}\}\}$, hence $\mathcal{SW}(DTA(G^{k,\epsilon})) = 2^k$. On the other hand, the maximum weight matching for $G^{k,\epsilon}$ (and sufficiently small $\epsilon$) is $M^*(G^{k,\epsilon}) = \{\{a_i, b_i\} \colon 0 \leq i \leq k+1\}$ with $\mathcal{SW}(M^*(G^{k,\epsilon})) = 2^{k+1} + 2^{k+2} - 2 - (k+2)\epsilon$. Hence,

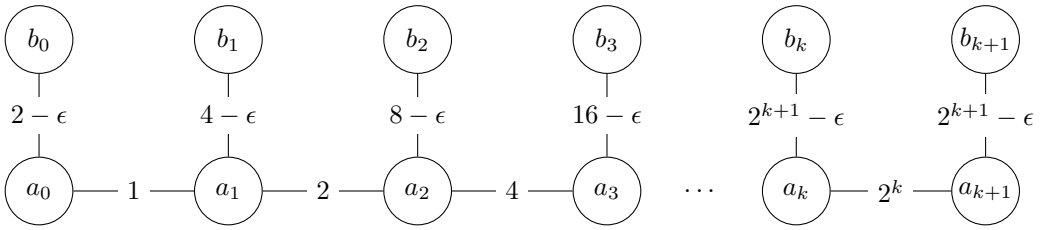



**Figure 3** Family of instances for tightness of competitive ratio in Theorem 11.

$$c_{DTA} = \inf_{G,\sigma} \frac{\mathcal{SW}(DTA(G,\sigma))}{\mathcal{SW}(M^*(G,\sigma))} \leq \inf_{k\geq 1,\epsilon>0} \frac{\mathcal{SW}(DTA(G^{k,\epsilon},\sigma^k))}{\mathcal{SW}(M^*(G^{k,\epsilon}))}$$
$$= \inf_{k\geq 1,\epsilon>0} \frac{2^k}{2^{k+1}+2^{k+2}-2-(k+2)\epsilon} = \frac{1}{6}.$$

Next, we show that $c_{DTA} \geq \frac{1}{6}$. Let $M$ be the matching produced by $DTA$ and let $M^*$ be a maximum weight matching. Without loss of generality, we may assume that all edges in $M^*$ have positive weight. Further let $F \subseteq E$ be the set of edges that were formed by $DTA$ at some point, i.e., $F$ consists of the edges in $M$ as well as all edges that have been dissolved. The key idea is to consider the relation induced by the replacement of edges. More precisely, given two edges $e, e' \in F$, we say that $e$ *dominates* $e'$ with respect to replacement, written as $e \succ_R e'$, if there exists a chain of edges $e_0, \dots, e_j$ such that $e_0 = e'$, $e_j = e$ and for $i \in [j]$, the formation of $e_i$ has lead to the dissolution of $e_{i-1}$. Note that $M$ consists precisely of the maximal elements in $F$ with respect to $\succ_R$.

We define a function $\mu\colon M^* \to 2^F$ as follows. Let $m = \{a, b\} \in M^*$ and assume that $b$ arrives after $a$. For a set of edges $F' \subseteq F$, we define $\max_{\succ_R}(F') = \{f \in F' \colon \nexists f' \in F \text{ with } f' \succ_R f\}$.

If, at the arrival of $b$, $a$ is already matched with $c$ and $2w(a,c) > w(a,b)$, then $\mu(m) = \max_{\succ_R}\{f \in F \colon f \succsim_R \{a,c\}, a \in f\}$. If, at the arrival of $b$, $a$ is already matched with $c$ and $2w(a,c) \leq w(a,b)$, then $b$ is matched with some $d$ and we define $\mu(m) = \max_{\succ_R}(\{f \in F \colon f \succsim_R \{a,c\}, a \in f\} \cup \{f \in F \colon f \succsim_R \{b,d\}, b \in f\})$. If at the arrival of $b$, $a$ is unmatched, then $b$ is matched with some $d$ and we define $\mu(m) = \max_{\succ_R}\{f \in F \colon f \succsim_R \{b,d\}, b \in f\}$. Note that $\mu$ always maps to a non-empty set. Indeed, if $a$ is unmatched or matched to an agent $c$ with $2w(a,c) \leq w(a,b)$, then $b$ will certainly be matched because matching with $a$ is an eligible option. Note that $\mu$ is a set-valued function, but for all $m \in M^*$, it holds that $|\mu(m)| \leq 2$. Moreover, the only case where $\mu(m)$ contains two edges is if we are in the second case and the edge $\{a,b\}$ is not created.

Bounding the weight of $M^*$ proceeds in three steps. First, we bound the weights of edges in $M^*$ by their image set in $F$. Then, we show that each edge in $F$ is only in the image of few edges. Lastly, we bound the weight accumulated by edges dominated with respect to $\succ_R$.

To illustrate the proof, it is useful to consider the example in Figure 3 from the first part of the proof. There, $F = \{\{a_i, a_{i+1}\} \colon 0 \leq i \leq k\}$. Moreover, the maximum weight matching is $M^* = \{\{a_i, b_i\} \colon 0 \leq i \leq k+1\}$. For $0 \leq i \leq k$ we have $\mu(\{a_i, b_i\}) = \{\{a_i, a_{i+1}\}\}$ according to the first case in the definition of $\mu$. In addition, $\mu(\{a_{k+1}, b_{k+1}\}) = \{\{a_k, a_{k+1}\}\}$, also according to the first case in the definition of $\mu$.

▷ Claim 12. For all $m \in M^*$, it holds that $w(m) \leq 2w(\mu(m))$.

Proof. Let $m = \{a, b\} \in M^*$ and assume that $b$ arrives after $a$. Assume first that, at the arrival of $b$, $a$ is already matched with $c$ and $2w(a, c) > w(a, b)$. Then, for every edge $f \in F$ with $f \succsim_R \{a, c\}$ and $a \in f$, there exists a sequence of edges $e_0, \dots, e_j$ such that $e_0 = e$, $e_j = f$ and for $i \in [j]$, $e_i$ has replaced $e_{i-1}$. Hence, $w(e) = w(e_0) \leq w(e_1) \leq \cdots \leq w(e_j) = w(f)$. It follows that $w(m) < 2w(a, c) \leq 2w(\mu(m))$.

Assume next that, at the arrival of $b$, $a$ is already matched with $c$ and $2w(a, c) \leq w(a, b)$. Then, $b$ will be matched with an agent $d$ such that $w(b, d) \geq w(a, b) - w(a, c)$. As in the first case, $w(a, c) \leq w(\max_{\succ_R}\{f \in F \colon f \succsim_R \{a, c\}, a \in f\})$ and $w(b, d) \leq w(\max_{\succ_R}\{f \in F \colon f \succsim_R \{b, d\}, b \in f\})$.

If $d \in \{a, c\}$, then $\{a, b\} \precsim_R f$ for all $f \in \mu(m)$ and $w(m) = w(a, b) \leq w(\mu(m))$. Otherwise, i.e., if $d \neq a$ and $d \neq c$, then $\{a, c\} \cap \{b, d\} = \emptyset$ and therefore also $\max_{\succ_R}\{f \in F \colon f \succsim_R \{a, c\}, a \in f\} \neq \max_{\succ_R}\{f \in F \colon f \succsim_R \{b, d\}, b \in f\}$. Hence, $w(m) \leq w(b, d) + w(a, c) \leq w(\mu(m))$.

Finally, assume that $a$ is unmatched. Then, $b$ will be matched with an agent $d$ such that $w(b, d) \geq w(a, b)$. As before, $w(b, d) \leq w(\max_{\succ_R}\{f \in F \colon f \succsim_R \{b, d\}, b \in f\})$, and we conclude $w(m) \leq w(b, d) \leq w(\mu(m)) \leq 2w(\mu(m))$. This completes the proof of the claim. ◁

Next we bound the number of edges that map to the same edge. Given an edge $e \in F$, define $\mu^{-1}(e) := \{m \in M^* \colon \mu(m) = e\}$.

▷ Claim 13. If $e \in M$, then $|\mu^{-1}(e)| \leq 2$. If $e \in F \setminus M$, then $|\mu^{-1}(e)| \leq 1$.

Proof. By definition of $\mu$, for every $m \in M^*$ and $m' \in \mu(m)$, $m' \cap m \neq \emptyset$. Let $e \in F$. Since $M^*$ is a matching, there can be at most one edge in $M^*$ with a non-empty intersection with each of the two endpoints of $e$. Hence, $|\mu^{-1}(e)| \leq 2$ and the first part of the claim holds.

Now, let $e = \{a, b\} \in F \setminus M$ and assume for contraction that there exist edges $m, m' \in M^*$ with $m \neq m'$, $e \cap m = \{a\}$, $e \cap m' = \{b\}$, and $e \in \mu(m) \cap \mu(m')$. Since, $e \notin M$, the edge $e$ was replaced during the algorithm by another edge $e'$. By definition of *DTA*, it holds that $e \cap e' \neq \emptyset$, say $e \cap e' = \{a\}$. Moreover, $e' \succ_R e$, contradicting that $e \in \mu(m)$. Hence, $|\mu^{-1}(e)| \leq 1$. ◁

It remains to bound the weight of replaced edges. For this, we introduce the following notation. For $e \in M$, define $F_e = \{e' \in F \colon e \succ_R e'\}$.

▷ Claim 14. For all $e \in M$, it holds that $w(F_e) \leq w(e)$.

Proof. Let $e \in M$. Since each edge can only replace a single other edge, $F_e$ is of the form $\{e_1, \dots, e_j\}$ for some $j \geq 0$ such that, for all $i \in [j-1]$, $e_i$ has replaced $e_{i+1}$, and $e$ has replaced $e_1$. Moreover, since a replacement only happens upon a sufficiently large weight improvement, we know that, for all $i \in [j-1]$, $w(e_i) \geq 2w(e_{i+1})$, and $w(e) \geq 2w(e_1)$. Hence, for all $i \in [j]$, it holds that $w(e_i) \leq 2^{-i}w(e)$. Hence, $w(F_e) = \sum_{i=1}^{j} w(e_i) \leq w(e)\sum_{i=1}^{j} 2^{-i} \leq w(e)$. ◁

Combining all three claims, we compute

$$w(M^*) = \sum_{m \in M^*} w(m) \overset{\text{Claim 12}}{\leq} \sum_{m \in M^*} 2w(\mu(m)) \overset{\text{Claim 13}}{\leq} 4\sum_{e \in M} w(e) + 2\sum_{e \in F \setminus M} w(e)$$
$$\overset{\text{Claim 14}}{\leq} 4\sum_{e \in M} w(e) + 2\sum_{e \in M} w(e) = 6w(M).$$ ◀

The next two lemmas let us apply *DTA* as an online coalition formation algorithm.

▶ **Lemma 15.** *Let $G = (N, w)$ be a complete weighted graph and $M^*$ a maximum weight matching of $G$. Then, $w(M^*) \geq \frac{1}{n} w(E^+)$, where $E^+ = \left\{e \in \binom{N}{2} : w(e) > 0\right\}$.*

▶ **Lemma 16.** *Let $ALG$ be a $c$-competitive algorithm for online matching. Then, $ALG$ is $\frac{c}{n}$-competitive for online coalition formation.*

**Proof.** Let $(G, \sigma)$ be an arbitrary instance of online coalition formation. Let $M^*$ be a maximum weight matching for the underlying weighted graph $G$, and let $\pi^*$ be a partition maximizing social welfare. Let $E^+ = \{e \in E : w(e) > 0\}$ be the set of positive edges. Then,

$$\mathcal{SW}(ALG(G, \sigma)) \geq c\, 2\, w(M^*) \geq c\, 2\, \frac{1}{n}\, w(E^+) \geq \frac{c}{n}\, \mathcal{SW}(\pi^*).$$

The first inequality holds because $ALG$ is a $c$-competitive algorithm in the matching domain. The second inequality follows from Lemma 15. The last inequality holds because twice the sum of positive edges is an upper bound for the social welfare of any partition. ◀

As a consequence, we can apply $DTA$ in the coalition formation domain to obtain a $\Theta(1/n)$-competitive algorithm under free dissolution.

▶ **Corollary 17.** *$DTA$ is $\frac{1}{6n}$-competitive for coalition formation in the deterministic model under free dissolution.*

## 6 Boundaries for Optimal Algorithms

In both of our models, we have found algorithms with a competitive ratio of $\Theta\left(\frac{1}{n}\right)$. This raises the question, whether it is possible to achieve even better algorithms, for instance, with a constant competitive ratio. While we leave the ultimate answer to this question open, in this section, we give some insight why the optimal algorithm may be hard to find. We start with upper bounds for the performance of any algorithm in our two settings.

▶ **Proposition 18.** *In the random arrival model, no online coalition formation algorithm achieves a competitive ratio of more than $\frac{1}{2}$.*

**Proof.** Let $ALG$ be any online coalition formation algorithm. Assume for contradiction that $ALG$ is $c$-competitive for some $c > \frac{1}{2}$.

First, note that for any $x > 0$, if the edge between the first two agents has a weight of $x$, then $ALG$ has to form a coalition of size 2. Otherwise, the competitive ratio is 0 in the instance where just these two agents arrive.

Now, we define a family of instances similar to the instance in Figure 1, with the difference that the agents in the sets $X$ and $Y$ also have positive utilities. Let $\epsilon > 0$ be a sufficiently small number and $k$ be a positive integer. Consider the ASHG given by $G^{k,\epsilon} = (N^{k,\epsilon}, w^{k,\epsilon})$, where $N^{k,\epsilon} = \{a, b\} \cup X \cup Y$ with $|X| = |Y| = k$. Hence, there are a total of $n = 2k+2$ agents. Utilities are given as $w^{k,\epsilon}(a, b) = 1$, $w^{k,\epsilon}(a, x_1) = w^{k,\epsilon}(b, y_1) = w^{k,\epsilon}(x_1, x_2) = w^{k,\epsilon}(y_1, y_2) = \epsilon$ for $x_1, x_2 \in X$ and $y_1, y_2 \in Y$. All other utilities are $-1$.

For $\epsilon$ sufficiently small, the social welfare is approximately maximized if the only non-singleton coalition is $\{a, b\}$, yielding a social welfare of 2.

Consider a random arrival order $\sigma$. If the first pair of agents are both from $X \cup \{a\} \times Y$ or $Y \cup \{b\} \times X$, then $\mathcal{SW}(ALG, \sigma) \leq n^2 \epsilon$.

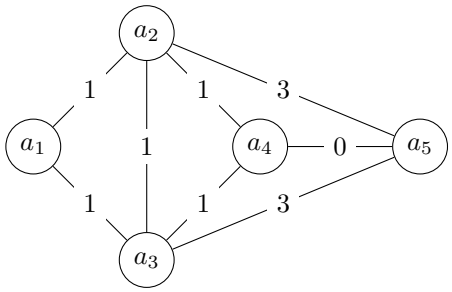



**Figure 4** Instance for online coalition formation algorithm with free dissolution. The two missing edges have a weight of $w(a_1, a_4) = w(a_1, a_5) = -12$.

In all other cases, i.e., for the other $2(k+1)^2$ possibilities for the first two agents, the social welfare is at most 2. Hence,

$$c_{ALG} \leq \inf_{k>0} \inf_{\epsilon>0} \frac{1}{2}\mathbb{E}_\sigma \left[\mathcal{SW}(ALG(G^{k,\epsilon}, \sigma))\right] \leq \inf_{k>0} \inf_{\epsilon>0} \frac{1}{2} \frac{4(k+1)^2 + 2(k+1)kn^2\epsilon}{(2k+2)(2k+1)}$$
$$= \inf_{k>0} \inf_{\epsilon>0} \frac{k+1}{2k+1} + \frac{kn^2}{2(2k+1)}\epsilon = \inf_{k>0} \frac{k+1}{2k+1} = \frac{1}{2}.$$

The infimum is attained in the limit for $k$ to infinity. ◀

▶ **Proposition 19.** *In the deterministic arrival model, no online coalition formation algorithm achieves a competitive ratio of more than $\frac{1}{3}$ under free dissolution.*

**Proof.** Let $ALG$ be any online coalition formation algorithm with free dissolution. Assume for contradiction that $ALG$ is $c$-competitive for some $c > \frac{1}{3}$.

Consider the ASHG given by $G = (N, w)$ depicted in Figure 4. The agents arrive in the order $(a_1, a_2, a_3, a_4, a_5)$. Since the algorithm is $c$-competitive for $c > \frac{1}{3}$, it has to maintain a coalition structure that has a social welfare of strictly more than $\frac{1}{3}$ of the current maximum social welfare. As a consequence the algorithm is forced to form the partitions $\{\{a_1, a_2\}\}, \{\{a_1, a_2, a_3\}\}$, and $\{\{a_1, a_2, a_3\}, \{a_4\}\}$ after $a_2$, $a_3$, and $a_4$ arrive, respectively. If the algorithm forms a different partition at any of these steps, then it achieves a social welfare of at most $\frac{1}{3}$ of the current maximum social welfare. Thus, the adversary can stop and then $c_{ALG} \leq \frac{1}{3}$, a contradiction.

Finally, agent $a_5$ arrives. In the ASHG given by $G$, the maximum social welfare of 18 is achieved by the partition $\pi^* = \{\{a_1\}, \{a_2, a_3, a_4, a_5\}\}$. However, the partition before $a_5$ arrives is $\{\{a_1, a_2, a_3\}, \{a_4\}\}$. It is easy to see that the highest social welfare that can be achieved with free dissolution is 6. This contradicts the assumption that $c_{ALG} > \frac{1}{3}$. ◀

A similar bound holds for the online matching setting. As we have already discussed in Section 5, our next result is a surprising contrast to the usual possibility of achieving a competitive ratio of 1/2 in many related settings [27, 20, 31]. Interestingly, once again, our construction only uses bipartite instances. Hence, the crucial property why we cannot achieve a better competitive ratio is that *all* agents arrive online.

▶ **Proposition 20.** *In the deterministic arrival model, no online matching algorithm achieves a competitive ratio of more than $\psi$ under free dissolution, where $\psi := \frac{2}{3+\sqrt{5}} \approx 0.382$.*

**Proof.** Let $ALG$ be any online coalition formation algorithm with free dissolution. Let $\psi = \frac{2}{3+\sqrt{5}}$ and assume for contradiction that $ALG$ is $c$-competitive for some $c > \psi$.

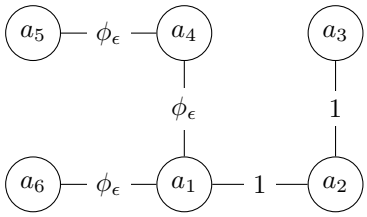



**Figure 5** Instance for online matching algorithm with free dissolution. We have $\phi_\epsilon = \frac{1+\sqrt{5}}{2} + \epsilon$ and the missing edges have weight 0. The figure depicts the case in the proof where $b = a_1$.

We provide an adversarial strategy along the game depicted in Figure 5. Since the algorithm is $c$-competitive, it has to maintain a coalition structure that has a social welfare of strictly more than a $c$-fraction of the current maximum social welfare.

First, two agents $a_1$ and $a_2$ arrive with $w(a_1, a_2) = 1$. Hence, $ALG$ has to form the edge $\{a_1, a_2\}$. Now, an agent $a_3$ arrives with $w(a_1, a_3) = 0$ and $w(a_2, a_3) = 1$. Then, $ALG$ only maintains a matching of positive weight if it leaves $a_3$ unmatched, or if it dissolves $\{a_1, a_2\}$ and forms $\{a_2, a_3\}$. Since the resulting situation is completely symmetric, we assume without loss of generality that $ALG$ leaves $a_3$ unmatched.

Let $\epsilon > 0$ and consider the constant $\phi_\epsilon := \frac{1+\sqrt{5}}{2} + \epsilon$.[5] Next, an agent $a_4$ arrives with $w(a_1, a_4) = \phi_\epsilon$ and $w(a_2, a_4) = w(a_3, a_4) = 0$. Now, $ALG$ only maintains a matching of positive weight if it leaves $a_4$ unmatched, or if it dissolves $\{a_1, a_2\}$ and forms $\{a_1, a_4\}$. In the former case, $ALG$ creates a matching of weight 1, while the maximum weight matching is $M_1 = \{\{a_1, a_4\}, \{a_2, a_3\}\}$ with $w(M_1) = 1 + \phi_\epsilon$. This implies that $c_{ALG} \leq \frac{1}{1+\phi_\epsilon} = \frac{1}{1+\frac{1+\sqrt{5}}{2}+\epsilon} < \frac{1}{1+\frac{1+\sqrt{5}}{2}} = \frac{2}{3+\sqrt{5}} = \psi < c$, a contradiction. Hence, $ALG$ has to dissolve $\{a_1, a_2\}$ and form $\{a_1, a_4\}$.

Next, an agent $a_5$ arrives with $w(a_4, a_5) = \phi_\epsilon$ and $w(a_1, a_5) = w(a_2, a_5) = w(a_3, a_5) = 0$. Similar to the situation after the arrival of the third agent, $ALG$ has to leave $a_4$ unmatched, or to dissolve $\{a_1, a_4\}$ and form $\{a_4, a_5\}$. Let $b \in \{a_1, a_5\}$, such that $ALG$ creates the matching $\{\{a_4, b\}\}$.

Now, an agent $a_6$ arrives with $w(b, a_6) = \phi_\epsilon$ and $w(x, a_6) = 0$ for all $x \in \{a_i \colon i \in [5]\} \setminus \{b\}$. Hence, $ALG$ will achieve a matching of weight at most $\phi_\epsilon$. For $b' \in \{a_1, a_4\}$ with $b' \neq b$, consider the matching $M_2 = \{\{a_2, a_3\}, \{b, a_6\}, \{b', a_4\}\}$. Then, $w(M_2) = 1 + 2\phi_\epsilon$. Hence,

$$c_{ALG} \leq \lim_{\epsilon \to 0} \frac{\phi_\epsilon}{1 + 2\phi_\epsilon} = \frac{\frac{1+\sqrt{5}}{2}}{1 + 1 + \sqrt{5}} = \frac{1 + \sqrt{5}}{2(2 + \sqrt{5})} = \frac{2}{3 + \sqrt{5}} = \psi < c.$$

This is our final contradiction, and hence such an algorithm cannot exist. ◀

Finally, we want to discuss general obstacles for finding classes of instances on which all algorithms perform poorly. Interestingly, the performance of $GDY$, $WGDY$, and $IWA$ is bounded by the same class of instances, namely the instances depicted in Figure 1. This raises the question whether this instance is a general worst-case instance. Our next result shows that this is not the case, unless there exists an algorithm that achieves a constant competitive ratio whenever the number of agents is known. Indeed, if there exists some highly valuable edge, then we can make use of an optimal stopping algorithm to achieve a good competitive ratio. To this end, we show how to apply the odds algorithm [11]. The details are discussed in the full version.

[5] To maintain games with rational weights, we can restrict attention to those $\epsilon$ where $\phi_\epsilon$ is rational.

▶ **Proposition 21.** *Let $I$ be a set of ASHGs and $\lambda \in (0, 1]$ a constant such that for each ASHG in $I$ given by $G = (N, w)$, the maximum weight edge $e_{\max}$ has a weight $w(e_{\max}) \geq \lambda \cdot \pi^*(G)$, where $\pi^*(G)$ maximizes social welfare. Then, there exists an online coalition formation algorithm $ALG$ with $c_{ALG} \in \Theta(1)$ on $I$ in the random arrival model with known $n$.*

## 7 Conclusion

We have considered two models of online coalition formation that both facilitate the existence of good algorithms compared to a deterministic arrival model. First, we have diminished the power of the adversary by considering a random arrival of agents. Second, we have increased the capabilities of an algorithm by allowing coalition dissolution. Both models allow for algorithms that achieve a competitive ratio of $\Theta(\frac{1}{n})$. Interestingly, this precisely gets rid of weight dependencies of the best algorithm in the deterministic model.

In both approaches, matchings play an important role. In the random arrival model, we present an algorithm whose output dominates the weight of a randomly created matching. Hence, matchings occur implicitly in the analysis of the algorithm. Under free dissolution, our coalition formation algorithm is itself a matching algorithm. The key challenge is to achieve a constant competitive ratio for online matching under the most general model, where all agents arrive online, and input graphs are weighted and possibly non-bipartite. The idea of our algorithm is to enhance the greedy algorithm by adding a threshold for the improvement in social welfare whenever dissolving a coalition (or edge).

Our work offers several promising directions for future research. First, while we have some indication that it is hard to obtain algorithms with a competitive ratio better than $\Theta(\frac{1}{n})$, we leave open the question whether there are algorithms obtaining a constant competitive ratio. Moreover, it would be interesting to find the optimal online matching algorithm for a fully online model with weighted non-bipartite input instances. Finally, there are many other classes of coalition formation games, such as fractional hedonic games [1] or ordinal classes of hedonic games. Considering a random arrival model or coalition dissolution for these games might lead to intriguing discoveries.

## References


1 H. Aziz, F. Brandl, F. Brandt, P. Harrenstein, M. Olsen, and D. Peters. Fractional hedonic games. *ACM Transactions on Economics and Computation*, 7(2):1–29, 2019.

2 H. Aziz, F. Brandt, and H. G. Seedig. Computing desirable partitions in additively separable hedonic games. *Artificial Intelligence*, 195:316–334, 2013.

3 S. Banerjee, H. Konishi, and T. Sönmez. Core in a simple coalition formation game. *Social Choice and Welfare*, 18:135–153, 2001.

4 V. Bilò, A. Fanelli, M. Flammini, G. Monaco, and L. Moscardelli. Nash stable outcomes in fractional hedonic games: Existence, efficiency and computation. *Journal of Artificial Intelligence Research*, 62:315–371, 2018.

5 V. Bilò, G. Monaco, and L. Moscardelli. Hedonic games with fixed-size coalitions. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, pages 9287–9295, 2022.

6 N. Boehmer, M. Bullinger, and A. M. Kerkmann. Causes of stability in dynamic coalition formation. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, pages 5499–5506, 2023.

7 A. Bogomolnaia and M. O. Jackson. The stability of hedonic coalition structures. *Games and Economic Behavior*, 38(2):201–230, 2002.

8 F. Brandt and M. Bullinger. Finding and recognizing popular coalition structures. *Journal of Artificial Intelligence Research*, 74:569–626, 2022.

**9** F. Brandt, M. Bullinger, and L. Tappe. Single-agent dynamics in additively separable hedonic games. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, pages 4867–4874, 2022.

**10** F. Brandt, M. Bullinger, and A. Wilczynski. Reaching individually stable coalition structures. *ACM Transactions on Economics and Computation*, 11(1–2):4:1–65, 2023.

**11** F. T. Bruss. Sum the odds to one and stop. *The Annals of Probability*, 28(3):1384–1391, 2000.

**12** M. Bullinger. Pareto-optimality in cardinal hedonic games. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 213–221, 2020.

**13** M. Bullinger. Boundaries to single-agent stability in additively separable hedonic games. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 26:1–26:15, 2022.

**14** M. Bullinger and W. Suksompong. Topological distance games. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, pages 5549–5556, 2023.

**15** R. Carosi, G. Monaco, and L. Moscardelli. Local core stability in simple symmetric fractional hedonic games. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 574–582, 2019.

**16** K. Cechlárová and A. Romero-Medina. Stability in coalition formation games. *International Journal of Game Theory*, 29:487–494, 2001.

**17** D. Dimitrov, P. Borm, R. Hendrickx, and S. C. Sung. Simple priorities and core stability in hedonic games. *Social Choice and Welfare*, 26(2):421–433, 2006.

**18** J. H. Drèze and J. Greenberg. Hedonic coalitions: Optimality and stability. *Econometrica*, 48(4):987–1003, 1980.

**19** E. Elkind, A. Fanelli, and M. Flammini. Price of pareto optimality in hedonic games. *Artificial Intelligence*, 288:103357, 2020.

**20** J. Feldman, N. Korula, V. Mirrokni, S. Muthukrishnan, and M. Pál. Online ad assignment with free disposal. In *Proceedings of the 5th International Conference on Web and Internet Economics (WINE)*, pages 374–385, 2009.

**21** M. Flammini, B. Kodric, G. Monaco, and Q. Zhang. Strategyproof mechanisms for additively separable and fractional hedonic games. *Journal of Artificial Intelligence Research*, 70:1253–1279, 2021.

**22** M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. On the online coalition structure generation problem. *Journal of Artificial Intelligence Research*, 72:1215–1250, 2021.

**23** M. Gairing and R. Savani. Computing stable outcomes in symmetric additively separable hedonic games. *Mathematics of Operations Research*, 44(3):1101–1121, 2019.

**24** Z. Huang, N. Kang, Z. G. Tang, X. Wu, Y. Zhang, and X. Zhu. How to match when all vertices arrive online. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 17–29, 2018.

**25** Z. Huang and T. Tröbst. Online matching. In F. Echenique, N. Immorlica, and V. V. Vazirani, editors, *Online and Matching-Based Market Design*. Cambridge University Press, 2023.

**26** C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 587–596, 2011.

**27** R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, 1990.

**28** M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 597–605, 2011.

**29** M. Olsen. Nash stability in additively separable hedonic games and community structures. *Theory of Computing Systems*, 45:917–925, 2009.

**30** S. C. Sung and D. Dimitrov. Computational complexity in additive hedonic games. *European Journal of Operational Research*, 203(3):635–639, 2010.

**31** Y. Wang and S. C. Wong. Two-sided online bipartite matching and vertex cover: Beating the greedy algorithm. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 1070–1081, 2015.

# On $k$-Means for Segments and Polylines

**Sergio Cabello**
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

**Panos Giannopoulos**
Department of Computer Science, City, University of London, UK

## Abstract

We study the problem of $k$-means clustering in the space of straight-line segments in $\mathbb{R}^2$ under the Hausdorff distance. For this problem, we give a $(1+\epsilon)$-approximation algorithm that, for an input of $n$ segments, for any fixed $k$, and with constant success probability, runs in time $O(n + \varepsilon^{-O(k)} + \varepsilon^{-O(k)} \cdot \log^{O(k)}(\varepsilon^{-1}))$. The algorithm has two main ingredients. Firstly, we express the $k$-means objective in our metric space as a sum of algebraic functions and use the optimization technique of Vigneron [40] to approximate its minimum. Secondly, we reduce the input size by computing a small size coreset using the sensitivity-based sampling framework by Feldman and Langberg [21, 22]. Our results can be extended to polylines of constant complexity with a running time of $O(n + \varepsilon^{-O(k)})$.

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

**Keywords and phrases** $k$-means clustering, segments, polylines, Hausdorff distance, Fréchet mean

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.28

**Related Version** *Full Version*: `https://arxiv.org/abs/2305.10922`

**Funding** *Sergio Cabello*: Research partially supported by the Slovenian Research Agency (P1-0297, J1-2452, N1-0218, N1-0285).

## 1 Introduction

The *$k$-means* clustering problem is as follows: Given a point set in a metric space, find a set of points, called *centers*, such that the sum of the squared distances from each input point to its closest center is minimized (over all possible choices of centers). It is a fundamental algorithmic problem with a ubiquitous role in data analysis in numerous application domains. As such, it has been studied extensively in geometric and general metric spaces, under various constraints on the objective and the choice of centers, and with a focus on complexity lower and upper bounds and the quality of the (approximate) solution [1, 2, 3, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 25, 28, 31, 34].

In geometric settings, almost all previous work involves clustering points in some low- or high-dimensional Euclidean space. Notable exceptions include the work on $k$-means clustering for lines with point centers [32], and the works on *$k$-center* [5] and *$k$-median* [6, 10, 19, 37] clustering for polygonal curves with respect to the Fréchet distance; for $k$-center, one seeks to minimize the maximum distance to the closest center, while for $k$-median, one seeks to minimize just the sum of the distances (instead of the sum of the squares) to the closest centers. In this paper, we consider the $k$-means problem in the space of segments and of polylines of constant complexity in the plane with respect to the Hausdorff distance.


## 1.1 Formalization of the problem

Let $(\mathcal{S}, d_H)$ be the metric space of closed straight-line segments in $\mathbb{R}^2$, where $d_H$ is the Hausdorff distance. Given a set $S$ of $n$ weighted segments, where each $s \in S$ has an associated positive weight $w_s \in \mathbb{R}_{>0}$, and for any $k$ segments $s_1, \ldots, s_k$ playing the role of "centers" of the clusters, we define the objective function

$$\text{cost}_S(\{s_1, \ldots, s_k\}) \;:=\; \sum_{s \in S} w_s \cdot \min\{d_H^2(s_1, s), \ldots, d_H^2(s_k, s)\}$$

and define the $k$-means problem as the problem of finding a minimizer, i.e., a set of segments $S^* = \{s_1^*, \ldots, s_k^*\}$ that minimizes the above cost. Note here that quite often we deal with *unweighted* input segments. However, for technical reasons (made clear later in our discussion) in order to incorporate coresets in our algorithm, we have to consider the more general case of weighted segments. Also note that we study the *continuous* version of the problem, where the solution segments can come from anywhere in $(\mathcal{S}, d_H)$. This is harder than the so-called *discrete* version, where the solution segments have to be selected among the input segments.

We also consider the $k$-means problem for polylines, each with a bounded number of segments, under the Hausdorff distance, where the definition of the problem is analogous.

We remark here on an interesting connection to the older and closely related concept of the *Fréchet mean* [24]. This is a generalization of the classic notion of mean or average to any abstract metric space. For a finite point set $P$ in a metric space $(\mathcal{M}, d)$, a Fréchet mean is any minimizer of the so-called *Fréchet variance* $\text{cost}_P(q) \;:=\; \sum_{p \in P} d^2(q, p)$, taken over all $q \in \mathcal{M}$. For Euclidean spaces, the Fréchet mean is the usual arithmetic mean. (Other usual means can be recovered as Fréchet means by considering other distances.) The Fréchet mean is a well-studied concept in Statistics and in Riemannian spaces, where sometimes it is known as Karcher mean, see [38] for a general, comprehensive treatment. Computing a Fréchet-mean is precisely the 1-means clustering problem while the $k$-means is the generalization where the cost of each cluster is given by the functional defining the Fréchet mean.

## 1.2 Results

Our main result is a $(1+\varepsilon)$-approximation algorithm for the $k$-means problem in $(\mathcal{S}, d_H)$. The algorithm runs in $O\left(\left(n + \varepsilon^{-16k+4-\eta} + \varepsilon^{-12k-3}\log^{4k+1}(\varepsilon^{-1})\right)(\log(1/\delta))\right)$ time, for any fixed $k$, any $\eta > 0$, and with success probability at least $1-\delta$ (the constant hidden in the $O$-notation depends on $\eta$ and $k$).

There are two main ingredients in our algorithm. For the first one, described in Section 2, we express the $k$-means objective in the space $(\mathcal{S}, d_H)$ as a sum of algebraic functions of constant description complexity. This algebraic approach allows us to use the optimization technique of Vigneron [40] for approximating the minimum. This is, to the best of our knowledge, the first application of this technique in the context of clustering. While this technique is very expensive when applied directly to the entire set of input segments, we can decrease the running time dramatically by combining it with coresets. This is the second ingredient of our algorithm, described in Sections 3 and 4 namely, we use the sensitivity framework of Feldman and Langberg [21, 22] to compute a small coreset of the input and then we apply the former algebraic approach to the coreset.

We then extend this result to polylines of description complexity at most $\ell = O(1)$; this is given in Section 5. In this context, each input polyline and each solution polyline has at most $\ell$ segments, but we may put in the solution polylines that are not part of the input. The running time becomes $O\left(\left(n + \varepsilon^{-O(k\ell)}\right)\log(1/\delta)\right)$.

As a side-result, in the full version of this paper, we consider the Fréchet mean (or 1-means) problem in a concrete example with two perpendicular segments that intersect at their centers. We show that even in this simple setting the set of Fréchet means is surprisingly complex. The optimum is attained in a 3-dimensional subset of the 4-dimensional parameter space needed to model the space of candidate segments. This example also prompts to the benefit of looking into an algebraic approach for the general setting.

## 1.3 Related work

For general metric spaces, $k$-means (as well as $k$-median) clustering is APX-hard (when $k$ is part of the input) [15, 26]. Several polynomial-time, constant factor approximation algorithms are known for both the continuous and discrete versions of the problem [1, 8]. For the discrete version, there even exist algorithms that achieve factors arbitrarily close to the lower bound [26] and run in FPT-time with respect to $k$ and the approximation error $\varepsilon$ [17].

The Euclidean $k$-means, where the input is a set of points in $\mathbb{R}^d$, is NP-hard for $d = 2$ [34] and APX-hard when $d = \omega(\log n)$ [2]. The problem admits EPTASs with respect to $k$ and $\varepsilon$ [31] and with respect to $d$ and $\varepsilon$ [11, 13].

As for the Fréchet mean, it has been considered for persistence diagrams [36, 39], point sets on the unit circle [7], and in the space of graphs [23, 30, 35], to name a few metric spaces far from the Euclidean setting.

## 1.4 Definitions and notation

For each point $p \in \mathbb{R}^2$, we use $x(p)$ and $y(p)$ for its two coordinates. Thus, $p = (x(p), y(p))$. For any two points $p, q \in \mathbb{R}^2$, we denote by $pq$ the segment with endpoints $p$ and $q$, and by $|pq|$ the Euclidean distance between them: $|pq|^2 = (x(p) - x(q))^2 + (y(p) - y(q))^2$. For simplicity we assume that all input segments have positive length.

Recall that the Hausdorff distance $d_H(A, B)$ between any two compact subsets $A, B \subset \mathbb{R}^2$ is defined by

$$d_H(A, B) = \max\Big\{\max_{a\in A}\min_{b\in B}|ab|,\ \max_{b\in B}\min_{a\in A}|ab|\Big\}.$$

Define $\delta(a, B) = \min_{b\in B}|ab|$ for the (directed) distance from a point $a$ to a closed set $B$. It is well known and easy to see that for any two segments $s_1 = a_1b_1$ and $s_2 = a_2b_2$ in $\mathcal{S}$

$$d_H(s_1, s_2) = \max\{\delta(a_1, s_2), \delta(b_1, s_2), \delta(a_2, s_1), \delta(b_2, s_1)\}. \tag{1}$$

# 2 An algebraic approach to $k$-means in $(\mathcal{S}, d_H)$

We use the following adaptation of the definition of a *nice* family of functions by Vigneron [40, Section 2.1]. Let $\mathcal{F} = \{f_i : \mathbb{R}^d \to \mathbb{R} \mid i \in I\}$ be a finite family of functions, where $I$ is some index set. We say that $\mathcal{F}$ is *nice* if there exists a constant $\lambda > d > 0$ such that:

- each $f_i \in \mathcal{F}$ is nonnegative and bounded;
- for each $f_i \in \mathcal{F}$, there exists a semialgebraic set $\operatorname{supp}(f_i) \subseteq \mathbb{R}^d$ and an algebraic function $g_i$ of degree at most $\lambda$ with $f_i(x) = g_i(x)$ for $x \in \operatorname{supp}(f_i)$ and $f_i(x) = 0$ for $x \notin \operatorname{supp}(f_i)$;
- for each $f_i \in \mathcal{F}$, the semialgebraic set $\operatorname{supp}(f_i) \subseteq \mathbb{R}^d$ is a boolean combination of at most $\lambda$ subsets of $\mathbb{R}^d$, each of them defined by an polynomial inequality of degree at most $\lambda$;
- for each $f_i \in \mathcal{F}$, the restriction of $f_i$ to $\operatorname{supp}(f_i)$ is continuous.

Note that the definition allows that the sets $\mathrm{supp}(f_i)$ are open, closed or mixed. It also allows that $f_i$ is discontinuous in $\mathbb{R}^d \setminus \mathrm{supp}(f_i)$, which may include the boundary of $\mathrm{supp}(f_i)$ in some cases.

Our use of this concept will be through the following result for computing an approximation to the minimum of the function $\sum_i f_i$.

▶ **Theorem 1** (Adaptation of Theorem 3.4 in Vigneron [40]). *Assume that $\varepsilon \in (0,1)$. Let $\mathcal{F} = \{f_i : \mathbb{R}^d \to \mathbb{R} \mid i \in I\}$ be a nice family of $m$ functions. Define $g = \sum_{i\in I} f_i$ and assume that $\min_{x\in\mathbb{R}^d} g(x)$ exists. Then we can compute a point $x'_\varepsilon \in \mathbb{R}^d$ such that $g(x'_\varepsilon) \leq (1+\varepsilon)\min_{x\in\mathbb{R}^d} g(x)$ in time $O(m^{2d-2+\eta} + (m/\varepsilon)^{d+1}\log^{d+1}(m/\varepsilon))$ for any $\eta > 0$.*[1] *The constant hidden in the $O$-notation depends on $\eta$ and on $d$.*

Let us first consider the simpler case of two segments and how their Hausdorff distance is defined. We parameterize a segment $ab$ as the point $(x(a), y(a), x(b), y(b))$ in $\mathbb{R}^4$. Note that in this parameterization we have the artifact that the segments $ab$ and $ba$ give different points in $\mathbb{R}^4$.

Let $\ell_{ab}$ be the line supporting a segment $ab$. For a point $p$, the distance $\delta(p, ab)$ is given by one of the three terms $|pa|$, $|pb|$, or $\delta(p, \ell_{ab})$. For a point $q \in \mathbb{R}^2$ and a segment $ab$, let $\ell_\perp(q, ab)$ be the line perpendicular to $\ell_{ab}$ through $q$. The lines $\ell_\perp(a, ab)$ and $\ell_\perp(b, ab)$ partition the plane into three 2-dimensional faces (Figure 1) with closures

$$\begin{aligned}
\sigma(ab) &= \text{the closed slab between } \ell_\perp(a, ab) \text{ and } \ell_\perp(b, ab),\\
\tau(a, ab) &= \text{the closed halfspace defined by } \ell_\perp(a, ab) \text{ that does not contain } b,\\
\tau(b, ab) &= \text{the closed halfspace defined by } \ell_\perp(b, ab) \text{ that does not contain } a.
\end{aligned}$$

We then have

$$\delta(p, ab) \;=\; \begin{cases} |pa| & \text{if } p \in \tau(a, ab),\\ |pb| & \text{if } p \in \tau(b, ab),\\ \delta(p, \ell_{ab}) & \text{if } p \in \sigma(ab). \end{cases}$$

From Equation (1), we conclude that, for any two segments $ab$ and $a'b'$, the distance $d_H(ab, a'b')$ is given by one of the functions in the family

$$\mathcal{F}(ab, a'b') \;:=\; \big\{|aa'|,\ |ab'|,\ |ba'|,\ |bb'|,\ \delta(a, \ell_{a'b'}),\ \delta(b, \ell_{a'b'}),\ \delta(a', \ell_{ab}),\ \delta(b', \ell_{ab})\big\}.$$

We next argue that all the expressions involved are algebraic. A point $p$ lies on the line $\ell_\perp(a, ab)$ if and only the scalar product of the vectors $\vec{ap}$ and $\vec{ab}$ is zero. This is equivalent to $\big(x(p), y(p), x(a), y(a), x(b), y(b)\big)$ being a zero of the algebraic (actually polynomial) function

$$\psi(x, y, x_a, y_a, x_b, y_b) := (x - x_a)(x_b - x_a) + (y - y_a)(y_b - y_a).$$

The sign of this expression also tells us which side of $\ell_\perp(a, ab)$ the point $p$ lies on. Note that this function is linear in $x$ and $y$, while it is quadratic in $x_a$ and $y_a$. Symmetrically, the sign of $\psi\big(x(p), y(p), x(b), y(b), x(a), y(a)\big)$ tells us which side of $\ell_\perp(b, ab)$ point $p$ lies on.

In the following, we will treat the segment $a'b'$ as variable, identified with $\mathbb{R}^4$, while the segment $ab$ will be fixed. We will show that the space $\mathbb{R}^4$ can be decomposed into cells such that, within a cell, the distance $d_H(ab, a'b')$ is defined always by the same function from

[1] As noted in Vigneron [40], one needs to assume either the Real-RAM model of computation (which is standard in computational geometry) or a model where we can choose the precision of the intermediate computations, and then the computational complexity of the algorithm depends on the desired precision.

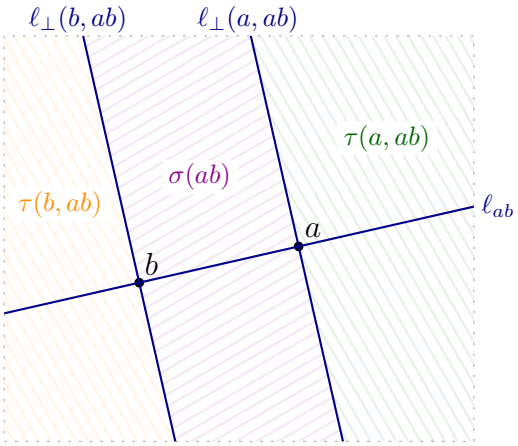



**Figure 1** The regions $\sigma(ab)$, $\tau(a, ab)$ and $\tau(b, ab)$.

$\mathcal{F}(ab, a'b')$. Such a decomposition is given by the eight algebraic hypersurfaces describing the conditions

$$a' \in \ell_\perp(a, ab),\ b' \in \ell_\perp(a, ab),\ a' \in \ell_\perp(b, ab),\ b' \in \ell_\perp(b, ab),$$
$$a \in \ell_\perp(a', a'b'),\ b \in \ell_\perp(a', a'b'),\ a \in \ell_\perp(b', a'b'),\ b \in \ell_\perp(b', a'b'),$$

together with a set of hypersurfaces, "bisectors", each defined by the set of points where two appropriate functions from $\mathcal{F}(ab, a'b')$ meet; this will become clear shortly. Finally, we note that each function in $\mathcal{F}(ab, a'b')$ is algebraic of constant degree; for example, elementary algebra shows that

$$\delta^2(a', \ell_{ab}) = \frac{\Big(\big(x(b) - x(a)\big)\big(y(a) - y(a')\big) - \big(x(a) - x(a')\big)\big(y(b) - y(a)\big)\Big)^2}{\big(x(a) - x(b)\big)^2 + \big(y(a) - y(b)\big)^2}.$$

We parameterize the space of (sequences of) $k$ segments $a_1b_1, \ldots, a_kb_k$ (the $k$ candidate cluster centers) by the point

$$\big(x(a_1), y(a_1), x(b_1), y(b_1), \ldots, x(a_k), y(a_k), x(b_k), y(b_k)\big) \in \mathbb{R}^{4k}.$$

Similarly, each $z \in \mathbb{R}^{4k}$ defines a $k$-tuple of segments with $s_1(z) = a_1(z)b_1(z), \ldots, s_k(z) = a_k(z)b_k(z)$ by taking the inverse of the parameterization.

▶ **Theorem 2.** *Let $k$ be a fixed, positive integer and let $s$ be a segment in the plane. In $O(1)$ time we can construct a nice family $\mathcal{F}_s = \{f : \mathbb{R}^{4k} \to \mathbb{R}\}$ of $O(1)$ functions such that*

$$\forall z \in \mathbb{R}^{4k} : \quad \sum_{f \in \mathcal{F}_s} f(z) = \min_{i \in [k]} d_H^2(s, s_i(z)).$$

**Proof.** Let $s = ab$ be the fixed segment. For each index $i \in [k]$, we consider the set $\Sigma(i)$ of 8 hypersurfaces in $\mathbb{R}^{4k}$, each of them given by one of the following conditions

$$a_i \in \ell_\perp(a, ab),\ b_i \in \ell_\perp(a, ab),\ a_i \in \ell_\perp(b, ab),\ b_i \in \ell_\perp(b, ab),$$
$$a \in \ell_\perp(a_i, a_ib_i),\ b \in \ell_\perp(a_i, a_ib_i),\ a \in \ell_\perp(b_i, a_ib_i),\ b \in \ell_\perp(b_i, a_ib_i).$$

Note that here $x(a)$, $y(a)$, $x(b)$ and $y(b)$ are input data while $x(a_i)$, $y(a_i)$, $x(b_i)$ and $y(b_i)$ are variables defining coordinates in the parameter space $\mathbb{R}^{4k}$.

Set $\Sigma := \cup_{i\in[k]}\Sigma(i)$ and let $\mathcal{A}_\Sigma$ be the arrangement in $\mathbb{R}^{4k}$ defined by $\Sigma$. From the foregoing discussion, we have the following property: for each cell $c$ of $\mathcal{A}_\Sigma$ and each index $i \in [k]$, the set of functions $\mathcal{F}(s, a_ib_i)$ stays the same and each of the distances $\delta(a, a_ib_i), \delta(b, a_ib_i), \delta(a_i, ab)$, and $\delta(b_i, ab)$, is given by the same function from $\mathcal{F}(s, a_ib_i)$. Thus, for all $z \in c$, the distance $d_H(s, s_i(z))$ is described by the maximum among the same four functions from $\mathcal{F}(s, a_ib_i)$.

In order to make clear that only the coordinates of $a_i$ and $b_i$ are relevant in the functions in $\mathcal{F}(s, a_ib_i)$, we change the notation to $\mathcal{G}_i$ and take each function $g$ of $\mathcal{G}_i$ to map from $\mathbb{R}^{4k}$ to $\mathbb{R}$. Formally, for each function $f \in \mathcal{F}(s, a_ib_i)$ we put into $\mathcal{G}_i$ the function $g(z) := f(ab, s_i(z))$.

We next define a set $\Lambda$ of algebraic hypersurfaces in $\mathbb{R}^{4k}$ playing the role of "bisectors". For each $i, j \in [k]$ with $i \le j$, we define $\Lambda(i, j)$ as the hypersurfaces given by equating each function of $\mathcal{G}_i$ to each function of $\mathcal{G}_j$. Note that each hypersurface is defined by a polynomial equality of degree at most 6. Since $\mathcal{G}_i$ has 8 functions for each $i \in [k]$, the set $\Lambda(i, j)$ has at most $8^2 = 64$ hypersurfaces (it is 32 for $\Lambda(i, i)$).

Set $\Lambda := \cup_{i\in[k]} \cup_{j\in[k],i\le j} \Lambda(i, j)$ and let $\mathcal{A}_\Lambda$ be the arrangement in $\mathbb{R}^{4k}$ induced by $\Lambda$. For each cell $c \in \mathcal{A}_\Lambda$ the sign of each function $g(z) - g'(z)$ remains constant for $g \in \mathcal{G}_i$, $g' \in \mathcal{G}_j$ and $z \in c$.

Finally, let $\mathcal{A}$ be the arrangement in $\mathbb{R}^{4k}$ induced by the hypersurfaces in $\Sigma \cup \Lambda$. Note that this is a refinement of $\mathcal{A}_\Sigma$ and $\mathcal{A}_\Lambda$, meaning that each cell of $\mathcal{A}$ is contained in a cell of $\mathcal{A}_\Sigma$ and a cell of $\mathcal{A}_\Lambda$.

Consider a cell $c \in \mathcal{A}$. Since $c$ is contained in a cell of $\mathcal{A}_\Sigma$, for each $i \in [k]$, each function in the set $\Delta_i(c) = \{\delta(a, s_i(z)), \delta(b, s_i(z)), \delta(a_i(z), ab)$, and $\delta(b_i(z), ab)\}$ is given by the same function of $\mathcal{G}_i$ for all $z \in c$. Moreover, since $c$ is contained in a cell of $\mathcal{A}_\Lambda$, for every two distinct functions $\delta, \delta' \in \Delta_i(c)$ the sign of $\delta - \delta'$ is constant for all $z \in c$. From these two facts we conclude that, for each $i \in [k]$, there is some function $g_{c,i}(z) \in \mathcal{G}_i$ such that $d_H(s, s_i(z)) = g_{c,i}(z)$ for all $z \in c$. This function can be easily determined in $O(1)$ time per cell by evaluating each function in $\Delta_i(c)$ at some arbitrary point in $c$.

Similarly, since $c$ is contained in a cell of $\mathcal{A}_\Lambda$, we have that for each distinct $i, j \in [k]$ the sign of

$$d_H(s, s_i(z)) - d_H(s, s_j(z)) \;=\; g_{c,i}(z) - g_{c,j}(z)$$

is constant for all $z \in c$. This implies that, for each cell $c \in \mathcal{A}$, there exists some index $\iota(c) \in [k]$ with the following property:

$$\forall j \in [k],\ z \in c: \quad d_H(s, s_{\iota(c)}(z)) \le d_H(s, s_j(z)).$$

In other words, the segment $s_{\iota(c)}(z)$ is a closest one to $s$ among $s_1(z), \ldots, s_k(z)$ and moreover the distance $d_H(s, s_{\iota(c)}(z))$ is given by a single function $g_{c,\iota(c)}$ from $\mathcal{G}_{\iota(c)}$. Thus, for each $z \in c$ it holds $\min_{i\in[k]} d_H(s, s_i(z)) = g_{c,\iota(c)}(z)$. As before, the function $g_{c,\iota(c)}(z)$ can be determined in $O(k)$ time per cell by evaluating each $d_H(s, s_i(z))$ at some arbitrary point in $c$.

For any set $A$, let $1_A$ be the function with $1_A(x) = 1$ if $x \in A$ and $1_A(x) = 0$ if $x \notin A$. For each cell $c \in \mathcal{A}$, define the function $h_c : \mathbb{R}^{4k} \to \mathbb{R}$ by $h_c(z) = 1_c(z) \cdot g^2_{c,\iota(c)}(z)$. Finally, set $\mathcal{F}_s := \{h_c \mid c \in \mathcal{A}\}$. We can then express the function

$$z \in \mathbb{R}^{4k} \mapsto \min_{i\in[k]} d^2_H(s, s_i(z))$$

as

$$\min_{i\in[k]} d^2_H(s, s_i(z)) \;=\; \sum_{c\in\mathcal{A}} 1_c(z)\, g^2_{c,\iota(c)}(z) \;=\; \sum_{c\in\mathcal{A}} h_c(z) \;=\; \sum_{h\in\mathcal{F}_s} h(z).$$

Since $\Sigma \cup \Lambda$ has $O(k^2) = O(1)$ hypersurfaces, the arrangement $\mathcal{A}$ has $O(O(k^2)^{4k}) = O(1)$ cells, each of them described by $O(k^2) = O(1)$ algebraic inequalities of constant description complexity and the family of functions $\mathcal{F}_s$ has the desired properties, where the constant $\lambda$ used to define the niceness is $O(k^{8k})$. Constructing $\mathcal{A}$ (i.e., with algebraic descriptions for each cell) takes $O(O(k^2)^{4k+1} 6^{O((4k)^4)}) = O(1)$ [4, Chapter 16]. The family $\mathcal{F}_s$ can be constructed in this time as well. ◀

We can now apply Theorem 1 combining all the functions $\mathcal{F}_s$ for $s \in S$ and compute a set of $k$ segments whose cost approximates that of an optimal set of segments.

▶ **Theorem 3.** *Let $k$ a fixed, positive integer and let $\varepsilon \in (0,1)$. Let $S$ be a family of $n$ segments in the plane with positive weights. We can compute $k$ segments $s_{1,\varepsilon}, \ldots, s_{k,\varepsilon}$ in $\mathbb{R}^2$ such that*

$$\text{cost}_S(\{s_{1,\varepsilon}, \ldots, s_{k,\varepsilon}\}) \;\leq\; (1+\varepsilon) \min\Big\{\text{cost}_S(\{s_1, \ldots, s_k\}) \mid s_1, \ldots, s_k \text{ segments}\Big\}$$

*in time $O(n^{8k-2+\eta} + (n/\varepsilon)^{4k+1} \log^{4k+1}(n/\varepsilon))$, for any $\eta > 0$. The constant hidden in the $O$-notation depends on $\eta$ and on $k$.*

**Proof.** For each segment $s \in S$ we compute the family $\mathcal{F}_s$ of Theorem 2. This takes $O(n) \cdot O(1) = O(n)$ time in total. To account for the weight $w_s > 0$ of the segment $s$, we replace in $\mathcal{F}_s$ each function $f \in \mathcal{F}_s$ with $w_s \cdot f$. Define $\mathcal{F} := \cup_{s \in S} \mathcal{F}_s$ and the function $g := \sum_{f \in \mathcal{F}} f$. Note that $\mathcal{F}$ is a family of $O(1) \cdot O(n) = O(n)$ nice functions and

$$\forall z \in \mathbb{R}^{4k} : \quad g(z) \;=\; \sum_{s \in S} \sum_{f \in \mathcal{F}_s} f(z) \;=\; \sum_{s \in S} w_s \cdot \min_{i \in [k]} d_H(s, s_i(z))^2 \;=\; \text{cost}_S(\{s_1(z), \ldots, s_k(z)\}).$$

We can then use Theorem 1 to find in time $O(|\mathcal{F}|^{2 \cdot 4k-2+\eta} + (|\mathcal{F}|/\varepsilon)^{4k+1} \log^{4k+1}(|\mathcal{F}|/\varepsilon))$, for any $\eta > 0$, a point $z'_\varepsilon \in \mathbb{R}^{4k}$ such that

$$g(z'_\varepsilon) \;\leq\; (1+\varepsilon) \min_{z \in \mathbb{R}^{4k}} \text{cost}_S(\{s_1(z), \ldots, s_k(z)\}).$$

The point $z'_\varepsilon \in \mathbb{R}^{4k}$ defines the segments $s_{1,\varepsilon} := s_1(z'_\varepsilon), \ldots, s_{k,\varepsilon} := s_k(z'_\varepsilon)$. As $s_1(z), \ldots, s_k(z)$ goes over all $k$ tuples of segments when $z$ iterates over all $\mathbb{R}^{4k}$, we have

$$\min_{z \in \mathbb{R}^{4k}} \text{cost}_S(\{s_1(z), \ldots, s_k(z)\}) \;=\; \min_{s_1, \ldots, s_k} \text{cost}_S(\{s_1, \ldots, s_k\}).$$

We conclude that

$$\text{cost}_S(\{s_{1,\varepsilon}, \ldots, s_{k,\varepsilon}\} \;=\; g(z'_\varepsilon) \;\leq\; (1+\varepsilon) \min_{s_1, \ldots, s_k} \text{cost}_S(\{s_1, \ldots, s_k\}).$$ ◀

## 3 A coreset for $k$-means in $(\mathcal{S}, d_H)$

We use the sensitivity framework of Feldman and Langberg [21, 22]. Let $F$ be a finite set of functions, each of them mapping from $\mathbb{R}^d$ to $\mathbb{R}_{\geq 0}$. The *sensitivity* of $f \in F$ with respect to $F$ is

$$\sigma(f, F) \;:=\; \sup_{z \in \mathbb{R}^d} \frac{f(z)}{\sum_{g \in F} g(z)}.$$

We also consider the following range space

$$\text{range}_{\geq}(F) \;:=\; \Big(F, \big\{\{f \in F \mid f(z) \geq r\} \mid z \in \mathbb{R}^d,\; r \in [0, \infty)\big\}\Big).$$

We will use the following theorem from [22], which we state here adapted to our needs.

▶ **Theorem 4** (Adaptation of Theorem 31 in Feldman et al. [22]). *Let $F$ be a set of $n$ functions from $\mathbb{R}^d$ to $[0,\infty)$ with the following properties:*

- *For each choice of weights $w_f > 0$ for $f \in F$, the range space $\text{range}_{\geq}(\{w_f \cdot f \mid f \in F\})$ has bounded VC-dimension.*
- *For each $f \in F$ we are given a value $\tilde{\sigma}(f)$ such that*

$$\tilde{\sigma}(f) \geq \frac{1}{|F|} \quad \text{and} \quad \tilde{\sigma}(f) \geq \sigma(f,F).$$

*Set $\tilde{\Sigma}(F) := \sum_{f\in F} \tilde{\sigma}(f)$. Let $\delta, \varepsilon$ be real values in $(0,1/2)$. In time $O(|F|)$ we can compute a subset $C \subseteq F$ of*

$$O\left(\frac{\tilde{\Sigma}(F)}{\varepsilon^2}\left(\log \tilde{\Sigma}(F) + \log\frac{1}{\delta}\right)\right)$$

*weighted functions and weights $u_f > 0$ for each $f \in C$ such that, with probability at least $1-\delta$:*

$$\forall z \in \mathbb{R}^d: \quad \left|\sum_{f\in F} f(z) - \sum_{f\in C} u_f \cdot f(z)\right| \leq \varepsilon \sum_{f\in F} f(z).$$

For each input segment $s \in S$, we define the function $f_s : \mathbb{R}^{4k} \to \mathbb{R}_{\geq 0}$ with

$$f_s(z) := \min\{d_H^2(s,s_1(z)),\ldots,d_H^2(s,s_k(z))\} = \left(\min\{d_H(s,s_1(z)),\ldots,d_H(s,s_k(z))\}\right)^2.$$

Here, the segments $s_1(z),\ldots,s_k(z)$ are the same that were used in the parameterization before Theorem 2. Set $F = \{f_s \mid s \in S\}$. In order to use the above theorem, we need appropriate sensitivity upper bounds $\tilde{\sigma}(f_s)$ for each $f_s \in F$ and a bound on the total sensitivity $\tilde{\Sigma}(F)$. Let $\text{opt}_k(S)$ be the cost of an optimal set of segments for $k$-means, i.e., $\text{opt}_k(S) = \min_{s_1,\ldots,s_k} \text{cost}_S(\{s_1,\ldots,s_k\})$.

▶ **Lemma 5.** *Let $s'_1\ldots,s'_{k'}$ be a bicriteria $(\alpha,\beta)$-approximation for $k$-means, that is, $k' \leq \beta k$ and $\text{cost}_S(\{s'_1,\ldots,s'_{k'}\}) \leq \alpha \cdot \text{opt}_k(S)$, where $\alpha,\beta \geq 1$. For each $i \in [k']$, let $S'_i$ be the segments of $S$ closer to $s'_i$ than to any other segment $s'_j$, $j \in [k'] \setminus \{i\}$; ties are solved arbitrarily so that $S'_1,\ldots,S'_{k'}$ is a partition of $S$. For each segment $s \in S$, let $\iota(s) \in [k']$ be such that $s \in S'_{\iota(s)}$. Define for each $s \in S$ the value*

$$\tilde{\sigma}(f_s) := \frac{32\alpha}{|S'_{\iota(s)}|} + \frac{16\alpha \cdot d_H^2(s,s'_{\iota(s)})}{\sum_{s'\in S'_{\iota(s)}} d_H^2(s',s'_{\iota(s)})} = \frac{32\alpha}{|S'_{\iota(s)}|} + \frac{16\alpha \cdot d_H^2(s,s'_{\iota(s)})}{\text{cost}_{S'_{\iota(s)}}(s'_{\iota(s)})}.$$

*Then $\tilde{\sigma}(f_s) \geq \sigma(f_s,F)$ and $\tilde{\sigma}(f_s) \geq \frac{1}{|F|}$.*

(The proof of the above lemma is technical and can be found in the full version of this paper.)

Finally, note that for the sensitivities $\tilde{\sigma}(f_s)$ defined in Lemma 5, we have the total sensitivity

$$\begin{aligned}
\tilde{\Sigma}(F) &= \sum_{s\in S} \tilde{\sigma}(f_s) = \sum_{s\in S}\left(\frac{32\alpha}{|S'_{\iota(s)}|} + \frac{16\alpha \cdot d_H^2(s,s'_{\iota(s)})}{\text{cost}_{S'_{\iota(s)}}(s'_{\iota(s)})}\right) \\
&= \sum_{i\in[k']}\left(\sum_{s\in S'_i}\frac{32\alpha}{|S'_i|} + \sum_{s\in S'_i}\frac{16\alpha \cdot d_H^2(s,s'_i)}{\text{cost}_{S'_i}(s'_i)}\right) = \sum_{i\in[k']}(32\alpha+16\alpha) \\
&= O(\beta\alpha k).
\end{aligned}$$

Next, we bound the VC-dimension of the range space associated with the input segments.

▶ **Lemma 6.** *Assume that we have a weight $w_s > 0$ for each $s \in S$ and consider the set of functions $F_w = \{w_s \cdot f_s \mid s \in S\}$. The range space* $\text{range}_{\geq}(F_w)$ *has VC-dimension* $O(1)$.

**Proof.** First note that the range space $\text{range}_{\geq}(F_w)$ is equivalent to the range space $(S, R)$, where the ranges are

$$\begin{aligned} R &= \big\{\{s \in S \mid (w_s \cdot f_s)(z) \geq r\} \mid z \in \mathbb{R}^{4k},\ r \in [0,\infty)\big\} \\ &= \big\{\{s \in S \mid w_s \cdot \min\{d_H^2(s, s_1(z)) \ldots, d_H^2(s, s_k(z))\} \geq r\} \mid z \in \mathbb{R}^{4k},\ r \in [0,\infty)\big\} \\ &= \big\{\{s \in S \mid \forall i \in [k] :\ \sqrt{w_s} \cdot d_H(s, s_i(z)) \geq \sqrt{r}\} \mid z \in \mathbb{R}^{4k},\ r \in [0,\infty)\big\}. \end{aligned}$$

Setting $w'_s = \sqrt{w_s}$ for each $s \in S$ and $r' = \sqrt{r}$, we get that the ranges are

$$R = \big\{\{s \in S \mid \forall i \in [k] :\ w'_s \cdot d_H(s, s_i(z)) \geq r'\} \mid z \in \mathbb{R}^{4k},\ r' \in [0,\infty)\big\}.$$

For each segment $s \in S$, consider the hypersurface $\lambda_s$ in $\mathbb{R}^{4k+1}$ given by the graph of the function $z \in \mathbb{R}^{4k} \mapsto w'_s \cdot d_H(s, s_i(z))$. This is $\lambda_s \equiv \big\{\big(z, w'_s \cdot d_H(s, s_i(z))\big) \in \mathbb{R}^{4k} \times \mathbb{R} \mid z \in \mathbb{R}^{4k}\big\}$. As it has been discussed and used in Section 2 when defining the set $\mathcal{F}(ab, a'b')$, the hypersurface $\lambda_s$ is contained in the union of 8 *algebraic* hypersurfaces of bounded degree, each of them being the graph of a function. Let $\Lambda_s$ be the set of those 8 algebraic hypersurfaces for the segment $s \in S$.

Set $\Lambda := \cup_{s \in S} \Lambda_s$ and let $\mathcal{A}$ be the arrangement in $\mathbb{R}^{4k+1}$ induced by $\Lambda$. Each point $(z, r') \in \mathbb{R}^{4k} \times \mathbb{R}$ gives a range to $R$, and two points in the same cell of $\mathcal{A}$ give exactly the same range to $R$ because, for each $s \in S$, the surface $\lambda_s$ is above, below or on all the points of the cell. It may happen that points in different cells of $\mathcal{A}$ give the same range, as one still has to check the condition $\forall i \in [k] :\ w'_s \cdot d_H(s, s_i(z)) \geq r'$. In any case, the number of cells in $\mathcal{A}$ is an upper bound to the number of ranges in $R$, which is exactly the number of ranges in $\text{range}_{\geq}(F_w)$.

Classical results in Real Algebraic Geometry imply that $\mathcal{A}$ has $|\Lambda|^{O(k)}$ cells; see for example [4, Chapter 7] or [33, Section 6.2]. This implies that the so-called shattering dimension of $\text{range}_{\geq}(F_w)$ is $O(k) = O(1)$. (See for example Har-Peled [27, Chapter 5] for the concept and the next property.) Since a range space has bounded shattering dimension if and only if it has bounded VC-dimension this implies that the VC-dimension of $\text{range}_{\geq}(F_w)$ is $O(1)$. The approach we have used is essentially an application of the methodology discussed by Matoušek [33, Section 10.3].

Note that in this proof we have not tried to optimize the bound on the VC-dimension because we assume $k$ is constant. Perhaps a better bound follows from adapting the result of Driemel at al. [20] to the case of weights. ◀

We can now apply Theorem 4 on $F$ to obtain the coreset.

▶ **Theorem 7.** *Assume that $k$ is a fixed positive integer. Let $\delta, \varepsilon$ be real values in $(0, 1/2)$. For any set $S$ of $n$ unweighted segments in the plane, we can compute in time $O(n \log(1/\delta))$ a subset $T \subseteq S$ of*

$$O\left(\varepsilon^{-2} \log \frac{1}{\delta}\right)$$

*segments and weights $u_s > 0$ for each $s \in T$ such that, with probability at least $1 - \delta$:*

$$\forall \text{ segments } s_1, \ldots, s_k :\ |\text{cost}_S(\{s_1, \ldots, s_k\}) - \text{cost}_T(\{s_1, \ldots, s_k\})| \leq \varepsilon \cdot \text{cost}_S(\{s_1, \ldots, s_k\}).$$

**Proof.** We first compute a bicriteria $(\alpha = O(1), \beta = O(1))$-approximation for $k$-means on $S$ by using the algorithm of Chen [9, Theorem A.4], which in turn is a modification of the algorithm by Indyk [29]. For a probability of error $\delta' = \delta/2$, the algorithm takes $O(n \log(1/\delta')) =$

$O(n\log(1/\delta))$ time and succeeds with probability at least $1-\delta'$ in finding a set $s_1,\ldots,s_{k'}$ of $k'=O(k)$ segments such that $\mathrm{cost}_S(\{s_1,\ldots,s_{k'}\}) \le O(1)\cdot\mathrm{cost}_S(\{s_1,\ldots,s_k\})$. Note that the algorithm of Chen is for the discrete version of $k$-means, where the centers under consideration must be a subset of $S$. However, it is well-known that the triangle inequality implies that this is a factor 4 off for the continuous $k$-means version. This factor 4 is then subsumed by the $O(1)$ approximation factor.

Let $F=\{f_s \mid s\in S\}$. We use the bicriteria approximation for the sensitivity upper bounds $\tilde{\sigma}(f_s)$, for each $f_s\in F$, as defined in Lemma 5. As discussed after Lemma 5, the total sensitivity $\tilde{\Sigma}(F)$ is $O(1)$ and, by Lemma 6, the VC-dimension of $\mathrm{range}_{\ge}(F)$ is $O(1)$. The result then follows using Theorem 4 with probability of error $\delta'=\delta/2$. The size of the set $C\subseteq F$ selected by Theorem 4 is $O(\tilde{\Sigma}(F)\varepsilon^{-2}(\log\tilde{\Sigma}(F)+\log(1/\delta'))=O(\varepsilon^{-2}\log(1/\delta))$, and each function $f\in C$ has a given weight $u_f>0$. We set $T=\{s\in S\mid f_s\in C\}$ and, for each segment $s\in T$, we define the weight $w_s:=u_{f_s}$.

With probability at least $1-\delta'$ we have

$$\forall z\in\mathbb{R}^{4k}: \quad \left|\sum_{f\in F} f(z) - \sum_{f\in C} u_f\cdot f(z)\right| \le \varepsilon\sum_{f\in F} f(z),$$

which can be rewritten as

$$\forall z\in\mathbb{R}^{4k}: \quad \left|\sum_{s\in S} f_s(z) - \sum_{s\in T} w_s\cdot f_s(z)\right| \le \varepsilon\sum_{s\in S} f_s(z).$$

Since $f_s(z)=\min\{d_H^2(s,s_1(z)),\ldots,d_H^2(s,s_k(z))\}$ and $s_1(z),\ldots,s_k(z)$ goes over all $k$ tuples of candidate segments when $z$ iterates over all $\mathbb{R}^{4k}$, the last statement is equivalent to

$$\forall \text{ segments } s_1,\ldots,s_k: \ |\mathrm{cost}_S(\{s_1,\ldots,s_k\})-\mathrm{cost}_T(\{s_1,\ldots,s_k\})| \le \varepsilon\cdot\mathrm{cost}_S(\{s_1,\ldots,s_k\}).$$

The algorithm may fail only if the bicriteria approximation of Chen fails or if the application of Theorem 4 fails, and each of them separately fails with probability at most $\delta/2$. ◀

## 4 Putting it all together

Let $S$ be a set of $n$ segments in the plane without weights. We first set a fixed probability of error $\delta=1/2$, which means that the terms $\log(1/\delta)$ become $O(1)$. We keep using $\varepsilon\in(0,1/2)$ as a parameter.

We first compute a weighted coreset $T\subseteq S$ with $|T|=O(\varepsilon^{-2}))$ elements in $O(n)$ time as described in Theorem 7; for each segment $s\in T$ we have a weight $w_s>0$. If $S^*=\{s_1^*,\ldots,s_k^*\}$ is an optimal set of segments for $S$, then from Theorem 7 we have that $\mathrm{cost}_T(S^*)\le(1+\varepsilon)\cdot\mathrm{cost}_S(S^*)$ with probability at least $1/2$.

We apply the $(1+\varepsilon)$-approximation algorithm of Theorem 3 on $T$, taking into account the weights of the segments. As $|T|=O(\varepsilon^{-2})$ the algorithm runs in time

$$O\left((\varepsilon^{-2})^{8k-2+\eta}+(\varepsilon^{-3})^{4k+1}\log^{4k+1}(\varepsilon^{-3})\right) = O\left(\varepsilon^{-16k+4-\eta}+\varepsilon^{-12k-3}\log^{4k+1}(\varepsilon^{-1})\right)$$

for any $\eta>0$. When $k=1$, the second summand dominates. When $k\ge 2$ and $\varepsilon$ is below some constant $\varepsilon_0$, the first summand dominates.

Let $T^* = \{t_1^*, \ldots, t_k^*\}$ be an optimal set of segments for $k$-means of the weighted set $T$. The algorithm of Theorem 3 has then provided a set $S_\varepsilon = \{s_{1,\varepsilon}, \ldots, s_{k,\varepsilon}\}$ of $k$ segments for which $\text{cost}_T(S_\varepsilon) \leq (1+\varepsilon) \cdot \text{cost}_T(T^*)$. Note that for the set $S_\varepsilon$ we also get from Theorem 7 that $(1-\varepsilon) \cdot \text{cost}_S(S_\varepsilon) \leq \text{cost}_T(S_\varepsilon)$. Since $\text{cost}_T(T^*) \leq \text{cost}_T(S^*)$, we conclude that

$$\begin{aligned}(1-\varepsilon) \cdot \text{cost}_S(S_\varepsilon) \;&\leq\; \text{cost}_T(S_\varepsilon) \;\leq\; (1+\varepsilon) \cdot \text{cost}_T(T^*) \;\leq\; (1+\varepsilon) \cdot \text{cost}_T(S^*) \\ &\leq\; (1+\varepsilon)^2 \cdot \text{cost}_S(S^*)\end{aligned}$$

or

$$\text{cost}_S(S_\varepsilon) \;\leq\; \frac{(1+\varepsilon)^2}{(1-\varepsilon)} \cdot \text{cost}_S(S^*) \;=\; (1+O(\varepsilon)) \cdot \text{cost}_S(S^*).$$

Setting $\varepsilon = \Theta(\varepsilon')$ appropriately, we get a $(1+\varepsilon')$-approximation for any desired $\varepsilon'$.

By independently repeating the algorithm $O(\log(1/\delta))$ times and taking the best among the solutions, we can reduce the probability of error to any given value $\delta$. Because $k = O(1)$, evaluating each candidate solution with respect to the whole set of segments takes $O(n)$ time. We summarize in the following.

▶ **Theorem 8.** *Let $k$ a fixed, positive integer and let $\delta, \varepsilon \in (0, 1/2)$. Let $S$ be a family of $n$ unweighted segments in the plane. We can compute $k$ segments $s_{1,\varepsilon}, \ldots, s_{k,\varepsilon}$ in $\mathbb{R}^2$ such that, with probability at least $1-\delta$,*

$$\text{cost}_S(\{s_{1,\varepsilon}, \ldots, s_{k,\varepsilon}\}) \;\leq\; (1+\varepsilon) \min_{s_1,\ldots,s_k} \text{cost}_S(\{s_1, \ldots, s_k\})$$

*in time $O\left(\left(n + \varepsilon^{-16k+4-\eta} + \varepsilon^{-12k-3} \log^{4k+1}(\varepsilon^{-1})\right)(\log(1/\delta))\right)$, for any $\eta > 0$.*

For $k = 1$, the running time is $O\left(\left(n + \varepsilon^{-15} \log^5(\varepsilon^{-1})\right)(\log(1/\delta))\right)$, while for $k \geq 2$ the running time is $O\left(\left(n + \varepsilon^{-16k+4-\eta}\right)(\log(1/\delta))\right)$ for any $\eta > 0$.

## 5 Extension to polylines

In this section we briefly discuss the extension of our result to the case of polylines of bounded complexity. To reduce the number of parameters, we assume that each polyline has at most $\ell$ segments and we search the $k$-means among polylines that have at most $\ell$ segments. (We can also handle the case where the input and the target centers have different complexities.) To simplify the discussion, we assume that each input polyline has exactly $\ell$ segments. We further assume that $\ell = O(1)$.

We regard each polyline $\pi$ as the union of segments and note that the distance between the polyline $\pi$ with segments $s_1, \ldots, s_\ell$ and the polyline $\pi'$ with segments $s'_1, \ldots, s'_\ell$ is

$$d_H(\pi, \pi') \;=\; \max\left\{\max_{i\in[\ell]} \min_{j\in[\ell]} d_H(s_i, s'_j),\; \max_{j\in[\ell]} \min_{i\in[\ell]} d_H(s'_j, s_i)\right\}.$$

Therefore the distance between any two polylines is described as a max-min combination of $O(\ell^2) = O(1)$ values.

A polyline with $\ell$ segments is parameterized by $2(\ell+1)$ real values. Therefore, a sequence of $k$ polylines with $\ell$ segments each is parameterized by a point in $\mathbb{R}^\kappa$ for $\kappa = 2k(\ell+1)$. (Before, for segments, we had $\kappa = 4k$.) Each $z \in \mathbb{R}^\kappa$ defines $k$ polylines $\pi_1(z), \ldots, \pi_k(z)$, each consisting of $\ell$ segments.

Let $\Pi$ be a set of polylines in the plane, each with $\ell$ segments. For each $\pi \in \Pi$, we define the function $f_\pi : \mathbb{R}^\kappa \to \mathbb{R}$ by

$$f_\pi(z) := \min\{d_H^2(\pi, \pi_1(z)), \ldots, d_H^2(\pi, \pi_k(z))\} = \big(\min\{d_H(\pi, \pi_1(z)), \ldots, d_H(\pi, \pi_k(z))\}\big)^2$$

and then define the set of functions $F = \{f_\pi \mid \pi \in \Pi\}$.

We first note that the VC-dimension of the range space $\text{range}_{\geq}(F_w)$ is $O(1)$, where $F_w$ is obtained from $F$ by scaling each $f_\pi \in F$ with a different scalar $w_\pi > 0$. The proof of Lemma 6 readily applies to this case as it only relies on the description complexity of $d_H(\pi, \pi_i(z))$ being constant, and each patch of the description being an algebraic function.

Next we note that we can use the bicriteria $(\alpha = O(1), \beta = O(1))$-approximation for $k$-means of Chen, as we did in the proof of Theorem 7. Indeed, this algorithm only requires that we can compute the distance between any two input objects, which we can do in constant time. The rest of the proof of Theorem 7 goes unchanged because Lemma 5 and Theorem 4 do not make any assumption related to segments beyond the VC-dimension. We thus obtain with probability at least $1/2$ a coreset $\tilde{\Pi}$ of $O(\varepsilon^{-2})$ input polylines, each of them with a positive weight $w_\pi$.

It remains to adapt Theorem 3 to the setting of polylines. As we have done in Theorem 2, for each polyline $\pi$ we can compute a family $\mathcal{F}_\pi$ of nice functions such that

$$f_\pi(z) = \sum_{f \in \mathcal{F}_\pi} f(z) = \min_{i \in [k]} d_H^2(\pi, \pi_i(z)) \quad \text{for all } z \in \mathbb{R}^\kappa.$$

Indeed, as we did in the proof of Theorem 2, we can break the parameter space $\mathbb{R}^\kappa$ using $O(k^2\ell^2) = O(1)$ algebraic hypersurfaces into $O(1)$ cells such that, within each cell, the max-max-min expression defining $d_H(\pi, \pi_i(z))$ is always the same algebraic expression. We can then apply Theorem 1 to the family of nice functions $\cup_{\pi \in \tilde{\Pi}} \mathcal{F}_\pi$, where each function in $\mathcal{F}_\pi$ has been scales with the corresponding weight $w_\pi$. Thus, we have an application of Theorem 1 in $\mathbb{R}^\kappa$ for $O(\varepsilon^{-2})$ functions. The running time is, for any $\eta > 0$,

$$\begin{aligned} O((\varepsilon^{-2})^{2\kappa-2+\eta} + (\varepsilon^{-3})^{\kappa+1} \log^{\kappa+1}(\varepsilon^{-3})) &= O(\varepsilon^{-4\kappa+4+\eta} + \varepsilon^{-3\kappa-3} \log^{\kappa+1}(\varepsilon^{-1})) \\ &= O(\varepsilon^{-O(k\ell)}). \end{aligned}$$

Like before, we can make $O(\log(1/\delta))$ independent repetitions to decrease the probability of failure to $\delta$. We summarize below.

▶ **Theorem 9.** *Let $k$ and $\ell$ be fixed, positive integers and let $\delta, \varepsilon \in (0, 1/2)$ be parameters. Let $\Pi$ be a family of $n$ unweighted polylines in the plane, each with at most $\ell$ segments. We can compute $k$ polylines $\pi_{1,\varepsilon}, \ldots, \pi_{k,\varepsilon}$ in $\mathbb{R}^2$, each with at most $\ell$ segments, such that, with probability at least $1 - \delta$,*

$$\text{cost}_\Pi(\{\pi_{1,\varepsilon}, \ldots, \pi_{k,\varepsilon}\}) \leq (1+\varepsilon) \min_{\pi_1, \ldots, \pi_k} \text{cost}_\Pi(\{\pi_1, \ldots, \pi_k\})$$

*in time $O\left(\left(n + \varepsilon^{-O(k\ell)}\right)(\log(1/\delta))\right)$.*

## References


1 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. *SIAM J. Comput.*, 49(4), 2020.

2 Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of Euclidean k-means. In *31st International Symposium on Computational Geometry, SoCG 2015*, volume 34 of *LIPIcs*, pages 754–767. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

3 Sayan Bandyapadhyay and Kasturi R. Varadarajan. On variants of k-means clustering. In *32nd International Symposium on Computational Geometry, SoCG 2016*, volume 51 of *LIPIcs*, pages 14:1–14:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

4 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer Berlin, Heidelberg, 2006.

5 Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating $(k, \ell)$-center clustering for curves. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2922–2938. SIAM, 2019.

6 Maike Buchin, Anne Driemel, and Dennis Rohde. Approximating $(k, \ell)$-median clustering for polygonal curves. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2697–2717, 2021.

7 Frédéric Cazals, Bernard Delmas, and Timothee O'Donnell. Fréchet mean and p-mean on the unit circle: Decidability, algorithm, and applications to clustering on the flat torus. In *19th International Symposium on Experimental Algorithms, SEA 2021*, volume 190 of *LIPIcs*, pages 15:1–15:16, 2021.

8 Deeparnab Chakrabarty, Maryam Negahbani, and Ankita Sarkar. Approximation algorithms for continuous clustering and facility location problems. In *30th Annual European Symposium on Algorithms, ESA 2022*, volume 244 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

9 Ke Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009.

10 Siu-Wing Cheng and Haoqiang Huang. Curve simplification and clustering under Fréchet distance. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 1414–1432. SIAM, 2023.

11 Vincent Cohen-Addad. A fast approximation scheme for low-dimensional $k$-means. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 430–440. SIAM, 2018.

12 Vincent Cohen-Addad, Hossein Esfandiari, Vahab S. Mirrokni, and Shyam Narayanan. Improved approximations for Euclidean $k$-means and $k$-median, via nested quasi-independent sets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1621–1628. ACM, 2022.

13 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. *J. ACM*, 68(6):44:1–44:34, 2021.

14 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

15 Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. On approximability of clustering problems without candidate centers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 2635–2648, 2021.

16 Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. Johnson coverage hypothesis: Inapproximability of k-means and k-median in $\ell_p$-metrics. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 1493–1530. SIAM, 2022.

17 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for $k$-means and $k$-median in Euclidean and minor-free metrics. *SIAM Journal on Computing*, 48(2):644–667, 2019.

18 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. Improved coresets and sublinear algorithms for power means in Euclidean spaces. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, pages 21085–21098, 2021.

19 Anne Driemel, Amer Krivosija, and Christian Sohler. Clustering time series under the Fréchet distance. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 766–785. SIAM, 2016.

20 Anne Driemel, André Nusser, Jeff M. Phillips, and Ioannis Psarros. The VC dimension of metric balls under Fréchet and Hausdorff distances. *Discret. Comput. Geom.*, 66(4):1351–1381, 2021.

21 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011*, pages 569–578. ACM, 2011.

22 Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for $k$-means, PCA, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657, 2020.

23 Daniel Ferguson and François G. Meyer. Computation of the sample Fréchet mean for sets of large graphs with applications to regression. In *Proceedings of the 2022 SIAM International Conference on Data Mining, SDM 2022*, pages 379–387, 2022.

24 Maurice Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. *Annales de L'Institut Henri Poincaré*, 10(4):215–310, 1948.

25 Fabrizio Grandoni, Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Rakesh Venkat. A refined approximation for Euclidean k-means. *Inf. Process. Lett.*, 176:106251, 2022.

26 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999.

27 Sariel Har-peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

28 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 291–300. ACM, 2004.

29 Piotr Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 428–434. ACM, 1999.

30 Eric D. Kolaczyk, Lizhen Lin, Steven J. Rosenberg, Jie Xu, and Jackson Walters. Averages of unlabeled networks: Geometric characterization and asymptotic behavior. *The Annals of Statistics*, 48(1):514–538, 2020.

31 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010.

32 Yair Marom and Dan Feldman. $k$-means clustering of lines for big data. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pages 12797–12806, 2019.

33 Jiří Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Graduate texts in mathematics*. Springer, 2002.

34 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984.

35 François G. Meyer. The Fréchet mean of inhomogeneous random graphs. In *Complex Networks & Their Applications X – Volume 1, Proceedings of the Tenth International Conference on Complex Networks and Their Applications, COMPLEX NETWORKS 2021*, volume 1015 of *Studies in Computational Intelligence*, pages 207–219, 2021.

36 Yuriy Mileyko, Sayan Mukherjee, and John Harer. Probability measures on the space of persistence diagrams. *Inverse Problems*, 27(12):124007, 2011.

37 Abhinandan Nath and Erin Taylor. $k$-median clustering under discrete Fréchet and Hausdorff distances. *J. Comput. Geom.*, 12:156–182, 2022.

38 Christof Schötz. *The Fréchet Mean and Statistics in Non-Euclidean Spaces*. PhD thesis, Heidelberg University, The Faculty of Mathematics and Computer Science, 2021.

39 Katharine Turner, Yuriy Mileyko, Sayan Mukherjee, and John Harer. Fréchet means for distributions of persistence diagrams. *Discret. Comput. Geom.*, 52(1):44–70, 2014.

40 Antoine Vigneron. Geometric optimization and sums of algebraic functions. *ACM Trans. Algorithms*, 10(1):4:1–4:20, 2014.

# Effective Resistances in Non-Expander Graphs

**Dongrun Cai**
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

**Xue Chen**
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

**Pan Peng**
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

## Abstract

Effective resistances are ubiquitous in graph algorithms and network analysis. For an undirected graph $G$, its effective resistance $R_G(s,t)$ between two vertices $s$ and $t$ is defined as the equivalent resistance between $s$ and $t$ if $G$ is thought of as an electrical network with unit resistance on each edge. If we use $L_G$ to denote the Laplacian matrix of $G$ and $L_G^\dagger$ to denote its pseudo-inverse, we have $R_G(s,t) = (\mathbf{1}_s - \mathbf{1}_t)^\top L^\dagger (\mathbf{1}_s - \mathbf{1}_t)$ such that classical Laplacian solvers [46] provide almost-linear time algorithms to approximate $R_G(s,t)$.

In this work, we study *sublinear* time algorithms to approximate the effective resistance of an *adjacent pair* $s$ and $t$. We consider the classical adjacency list model [41] for local algorithms. While recent works [4, 40, 31] have provided sublinear time algorithms for *expander graphs*, we prove several lower bounds for *general graphs* of $n$ vertices and $m$ edges:

1. It needs $\Omega(n)$ queries to obtain 1.01-approximations of the effective resistance of an adjacent pair $s$ and $t$, even for graphs of degree at most 3 except $s$ and $t$.
2. For graphs of degree at most $d$ and any parameter $\ell$, it needs $\Omega(m/\ell)$ queries to obtain $c\cdot\min\{d,\ell\}$-approximations where $c > 0$ is a universal constant.

Moreover, we supplement the first lower bound by providing a sublinear time $(1+\epsilon)$-approximation algorithm for graphs of *degree 2* except the pair $s$ and $t$.

One of our technical ingredients is to bound the expansion of a graph in terms of the smallest non-trivial eigenvalue of its Laplacian matrix after removing edges. We discover a new lower bound on the eigenvalues of perturbed graphs (resp. perturbed matrices) by incorporating the effective resistance of the removed edge (resp. the leverage scores of the removed rows), which may be of independent interest.

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

**Keywords and phrases** Sublinear Time Algorithm, Effective Resistance, Leverage Scores, Matrix Perturbation

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.29

**Related Version** *Full Version*: https://arxiv.org/abs/2307.01218

**Funding** *Pan Peng*: Supported in part by NSFC grant 62272431 and "the Fundamental Research Funds for the Central Universities".

**Acknowledgements** We thank Jingcheng Liu (Nanjing University) for suggesting the proof of Lemma 5 using the characteristic polynomial method. Also, we thank all anonymous reviewers for the helpful comments.


# 1 Introduction

Effective resistance of a graph is a fundamental quantity to measure the similarity between two vertices. Given an unweighted graph $G$ and two vertices $s$ and $t$, the $s$-$t$ effective resistance, denoted by $R_G(s,t)$, is defined as the electrical distance between $s$ and $t$ when $G$ represents an electrical circuit with each edge $e$ a resistor with electrical resistance 1. Together with the related concept electrical flows, effective resistances have played important roles in advancing the development of graph algorithms. They have been utilized for computing and approximating maximum flow [10, 34], generating random spanning tree [35, 43], designing faster algorithms for multicommodity flow [28], and graph sparsification [45, 15]. In addition, effective resistances have also found applications in machine learning and social network analysis. For example, it has been used for graph convolutional networks [2], for measuring the similarity of vertices in social networks [30, 40] and measuring robustness of networks [19].

Let $L_G$ denote the Laplacian matrix of $G$ and $L_G^\dagger$ denote its pseudo-inverse. Then the $s$-$t$ effective resistance admits an elegant expression $R_G(s,t) = (\mathbf{1}_s - \mathbf{1}_t)^\top \cdot L_G^\dagger \cdot (\mathbf{1}_s - \mathbf{1}_t)$ where $\mathbf{1}_u \in \mathbb{R}^n$ denotes the indicator vector at vertex $u$. Hence classical Laplacian solvers [46, 13] provide *almost-linear* time algorithms to approximate $R_G(s,t)$.

It has recently received increasing interest of designing *sublinear-time* (or local) algorithms for estimating effective resistances. In this setting, we are given query access to a graph and any specified vertex pair $s$ and $t$, our goal is to find a good approximation of the $s$-$t$ effective resistance, by making as few queries as possible. Such algorithms are particularly motivated by the ubiquitousness of modern massive graphs, on which traditional polynomial-time algorithms are no longer feasible. Andoni et al. [4] gave an algorithm that $(1+\epsilon)$-approximates $R_G(s,t)$ in $O(\frac{1}{\epsilon^2}\mathsf{poly}\log\frac{1}{\epsilon})$ time for $d$-regular expander graphs. Peng et al. [40] then generalized this algorithm to unbounded-degree expander graphs with an additive error $\epsilon$ and similar query complexity. Li and Sachdeva [31] then gave one algorithm that $(1+\epsilon)$-approximates the $s$-$t$ effective resistance in $O(\frac{\mathsf{poly}(\log n)}{\epsilon})$ time on expander graphs (which is implicit in the proof of Theorem 3.1 in [31]). However, the question of whether it is possible to obtain sublinear-time estimation for effective resistances on non-expander or general graphs remains largely open.

Besides the aforementioned sublinear-time algorithms for effective resistances on expander graphs, one can observe an $\Omega(n)$-query lower bound for approximating the effective resistance of *non-adjacent* pair $s$ and $t$. Indeed, consider an $n$-vertex path, on which the $s$-$t$ effective resistance is equivalent to the $s$-$t$ shortest path length. Intuitively, for the latter problem, any algorithm with a constant approximation ratio needs to well estimate the number of edges on the path from $s$ to $t$, which takes $\Omega(n)$ queries in the worst case.

In this paper, we consider the power and limitations of sublinear-time algorithms for $s$-$t$ effective resistance such that $s$ and $t$ are *adjacent*, i.e., $(s,t) \in E(G)$. The adjacency case is already interesting in many applications. For example, in the seminal work on graph sparsification [45], it suffices to have good estimations of the effective resistances between the endpoints of edges, i.e., the adjacent pairs. It is also known that the effective resistance multiplied by the edge weight is equal to the probability that the edge belongs to a randomly generated spanning tree (see e.g. [18]), which has found applications in random spanning tree generation.

On the other hand, for an adjacency pair, the lower bound from the previous discussion does not hold any more, as for any pair $s$ and $t$ such that $(s,t) \in E$, their effective resistance is exactly 1 on a path. A priori, it could be true that a $(1+\epsilon)$-approximation of the $s$-$t$ effective resistance for an adjacent pair can be found in sublinear time.

Now we state our results on sublinear-time algorithms for estimating the effective resistances. We will focus on unit-weighted graphs and the adjacency list model [41], in which the local algorithms can perform degree, neighbor queries and also sample vertices uniformly at random.

**Strong Lower Bounds.** First, we provide a strong lower bound for graphs of degree at most 3 except the given pair $s$ and $t$. For convenience, for a parameter $C > 1$, we say that value $a$ is a $C$-approximation of value $b$ if $a \in [b/C, b]$. In the following, we always assume that the given pair $s$ and $t$ are *adjacent*, i.e., $(s,t) \in E$.

▶ **Theorem 1.** *There are infinitely many $n$ and graphs of $n$ vertices such that any local algorithm with success probability 0.6 and approximation ratio 1.01 on $R_G(s,t)$ needs $\Omega(n)$ queries. This holds even for graphs whose vertices are of degree at most 3 except the adjacent pair $s$ and $t$.*

Next, observe that the trivial algorithm outputting 1 directly gives an approximation factor $\frac{2}{1/d(s)+1/d(t)}$ where $d(s)$ and $d(t)$ are the degrees of the adjacent pair $s$ and $t$. This is because $R_G(s,t) \geq \frac{1/d(s)+1/d(t)}{2}$ from the spectral graph theory [33]. This factor equals $d$ if the graph is regular and is between $\min\{d(s), d(t)\}$ and $2\min\{d(s), d(t)\}$ in general. If we consider large approximation factors instead of the $(1+\epsilon)$-approximation shown in Theorem 1, a natural question is can we design sublinear time algorithms that improve upon this trivial approximation ratio $\frac{2}{1/d(s)+1/d(t)}$? Our next theorem shows that it would take $\Omega(n)$ queries to improve this ratio significantly.

▶ **Theorem 2.** *There exist a universal constant $c > 0$ and infinitely many $n$ such that given any $d \geq 4$ and any $\ell \geq 4$, for graphs of $n$ vertices and degree at most $d$, any local algorithm to approximate $R_G(s,t)$ with success probability 0.6 and approximation ratio $1 + c \cdot \min\{d, \ell\}$ needs $\Omega(nd/\ell)$ queries.*

If we set $\ell = d$, this indicates that it takes $\Omega(n)$ queries to obtain an approximation ratio $o(d)$ for $d$-regular graphs since $m = nd/2$. In contrast, the corresponding trivial approximation ratio for adjacent pairs on $d$-regular graphs is $d$. If we fix $\ell$ and consider graphs with sufficiently large degree $d > \ell$, Theorem 2 also gives a trade-off between the query complexity and approximation ratio. Next we remark that we could incorporate additive errors into the lower bounds of Theorem 1 and Theorem 2. This is because these two bounds consider the task of distinguishing between graphs with $R_G(s,t) = 1$ and graphs with $R_G(s,t) < 1$.

On the other hand, for graphs of degree at most 2 except the given pair $s$ and $t$, we provide a $(1+\epsilon)$-approximation algorithm of sublinear time in Section 5.

**Total effective resistance.** Another important measure of a network is the total effective resistance, which is defined as $R_{tot}(G) = \sum_{u<v} R_G(u,v)$ and is also known as the Kirchhoff index of a graph (see e.g. [20, 19, 30] for numerous applications of $R_{tot}(G)$). Because of the elegant expression $R_{tot}(G) = n \cdot \sum_{i>1} \frac{1}{\lambda_i(L_G)}$ where the sum is over all non-trivial eigenvalues of the Laplacian of $G$, one may wonder whether there is a simpler approximation algorithm for the total effective resistance or not. However, we show that even for simple graphs with degree 2, any algorithm with approximation ratio $< 2$ needs $\Omega(n)$ queries.

▶ **Theorem 3.** *For any $n$ and any $\ell > 1$, any local algorithm for computing the total effective resistance with success probability 0.6 and approximation ratio $\ell$ needs $\Omega(n)$ queries. In particular, this holds even for $\ell = 2 - o(1)$ and graphs of degree at most 2.*

**Eigenvalues of graph perturbation.** One technical ingredient in the proof of Theorem 1 is to show that an expander graph is still an expander after removing one edge. A natural approach would be to compare the smallest non-trivial eigenvalue of the Laplacian of the perturbed graph to the corresponding one of the original graph. Since deleting one edge in $G$ does not change its edge expansion too much, one may use Cheeger's inequality (with other properties) to lower bound the new eigenvalue in terms of the original one. However, our key technical lemma provides a more direct bound by incorporating the effective resistance of the moved edge.

▶ **Lemma 4.** *Given a graph $G = (V, E)$ with $n$ vertices, let $\lambda_1(G) \leq \cdots \leq \lambda_n(G)$ be the eigenvalues of its Laplacian $L_G$. Given any edge $(u, v)$ in $G$, let $G' = \big(V, E \setminus \{(u, v)\}\big)$ be the graph obtained from $G$ by removing the edge $(u, v)$ and let $\lambda_1(G') \leq \ldots \leq \lambda_n(G')$ be the eigenvalues of its Laplacian $L_{G'}$. Then it holds that*

$$\forall i \in [n], \lambda_i(G') \geq \big(1 - R_G(u, v)\big) \cdot \lambda_i(G).$$

This lemma shows that after removing edge $(u, v)$ in an expander $G$ with $R_G(u, v)$ strictly less than 1, $G$ is still an expander. We remark that this requirement on the effective resistance $R_G(u, v)$ is necessary. Because there exists an expander graph $G$ with edges of effective resistance 1 and vertices of degree 1 (see Claim 10 in Section 3), one can not delete an arbitrary edge in $G$ while maintaining the expansion property.

We could consider the general problem of bounding the eigenvalues of a matrix after removing a few rows. Specifically, given a matrix $A \in \mathbb{R}^{m \times n}$ and $k$ distinct rows $\mathbf{a}_{\ell_1}, \ldots, \mathbf{a}_{\ell_k}$, the question is to compare all eigenvalues of $(A')^\top A'$, where $A' \in \mathbb{R}^{(m-k) \times n}$ removes row $\mathbf{a}_{\ell_1}, \ldots, \mathbf{a}_{\ell_k}$ from $A$, to the eigenvalues of $A^\top A$. Since $(A')^\top A' \preceq A^\top A$, each eigenvalue $\lambda_i\big((A')^\top A'\big) \leq \lambda_i\big(A^\top A\big)$. We give a lower bound for every $\lambda_i\big((A')^\top A'\big)$ by incorporating the leverage scores of those rows.

The (statistical) leverage scores of a matrix $A \in \mathbb{R}^{m \times n}$ provide a nonuniform importance sampling distribution over the $m$ rows of $A$, which plays a crucial role in randomized matrix algorithms (see [47] for a list of applications). For each row $\mathbf{a}_i$ in $A$, its leverage score $\tau_i$ is defined to be $\mathbf{a}_i^\top \cdot (A^\top A)^\dagger \cdot \mathbf{a}_i$. In fact, the leverage scores of a matrix are the analogues of the effective resistances of a graph [16]. More formally, let $B \in \mathbb{R}^{m \times n}$ denote the edge-incidence matrix of a graph $G$ as follows: Each edge $(u, v)$ of $G$ gives a row $\mathbf{1}_u - \mathbf{1}_v$ in $B$. Furthermore, the Laplacian matrix $L$ of $G$ equals $B^\top B$. If a row of $B$ corresponds to edge $(u, v)$, its leverage score $(\mathbf{1}_u - \mathbf{1}_v)^\top \cdot (B^\top B)^\dagger \cdot (\mathbf{1}_u - \mathbf{1}_v) = (\mathbf{1}_u - \mathbf{1}_v)^\top \cdot L^\dagger \cdot (\mathbf{1}_u - \mathbf{1}_v)$ turns out to be the effective resistance of $(u, v)$ in $G$.

Now we state Lemma 5 for the general problem. So Lemma 4 is a direct corollary of Lemma 5 by setting $A$ to the incidence matrix of $G$ as discussed above.

▶ **Lemma 5.** *Given a matrix $A \in \mathbb{R}^{m \times n}$, let $\lambda_1 \leq \cdots \leq \lambda_n$ be the eigenvalues of $A^\top A$. Moreover, for each $\ell \in [m]$, let $\mathbf{a}_\ell$ be row $\ell$ of $A$ and $\tau_\ell = \mathbf{a}_\ell^\top (A^\top A)^\dagger \mathbf{a}_\ell$ be its leverage score.*

*For any $k$ distinct indices $\ell_1, \ldots, \ell_k \in [m]$, let $A' \in \mathbb{R}^{(m-k) \times n}$ be the matrix obtained from $A$ by removing the corresponding $k$ rows $\mathbf{a}_{\ell_1}, \ldots, \mathbf{a}_{\ell_k}$; and let $\lambda'_1 \leq \cdots \leq \lambda'_n$ be the eigenvalues of $(A')^\top A'$. It holds that*

$$\forall i \in [n], \lambda'_i \in \big[(1 - \tau_{\ell_1} - \tau_{\ell_2} - \cdots - \tau_{\ell_k}) \cdot \lambda_i, \lambda_i\big].$$

## 1.1 Related Work

Previous research has studied the problem of how to quickly compute and approximate the effective resistances in the regime of polynomial-time algorithms, as such algorithms can be used as a crucial subroutine for other graph algorithms. For example, for any two vertices $s$

and $t$ in a $n$-vertex $m$-edge graph, one can $(1+\epsilon)$-approximate the $s$-$t$ effective resistance in $\tilde{O}(m+n\epsilon^{-2})$ [18] and $\tilde{O}(m\log(1/\epsilon))$ [13] time, respectively. To $(1\pm\epsilon)$-approximate the effective resistances between $s$ given pairs, Chu et al. [11] provided an algorithm in time $O(m^{1+o(1)}+(n+s)n^{o(1)}\cdot\epsilon^{-1.5})$. There are also algorithms that find $(1+\epsilon)$-approximations to the effective resistance between every pair of vertices in $\tilde{O}(n^2/\epsilon)$ time [27]. In order to exactly compute the $s$-$t$ or all-pairs effective resistance, the current fastest algorithms run in times $O(n^\omega)$ (by using the fastest matrix inversion algorithm [8, 24]), where $\omega<2.373$ is the matrix multiplication exponent [3]. Faster algorithms are known for planar graphs by using the nested dissection method [32].

There exists a line of works on how to efficiently maintain the effective resistances *dynamically* [42, 1, 21, 22, 17, 9], i.e., if the graph undergoes edge insertions and/or deletions, and the goal is to support the update operations and query for the effective resistances as quickly as possible, rather than having to recompute it from scratch each time.

For the total effective resistance, Ghosh et al. [20] studied algorithms for allocating edge weights on a given graph in order to minimize the total effective resistance. Li and Zhang [30] used Kirchhoff index (i.e., total effective resistance) as the measure of edge centrality in weighted networks and gave efficient algorithms for the measure.

Matrix perturbation considers the eigenvalues (singular values) of $A$ after adding a matrix $E$ of the same order. Various bounds on the error of matrix perturbation, such as absolute errors and relative errors, has been studied. We refer to the survey [26] and the reference therein for a complete overview. Even though Lemma 5 is an instantiation of Theorem 2.8 in [26] with (statistical) leverage scores, we provide two simpler proofs in this work which are more intuitive. Also, different perturbation bounds in terms of the leverage scores were provided in [25, 23].

Leverage scores, analogue to effective resistances of graphs, have wide applications in randomized matrix algorithms and large-scale data algorithms. The most notable property is that sampling the rows of a matrix $A$ via its leverage scores gives an efficient construction of the subspace embedding of $A$ [45]. This fact is extremely useful in designing ultra-efficient algorithms for linear regression and low rank approximations [12]. We refer to the survey [47] for a list of applications. Since bounding eigenvalues are sufficient for $\ell_2$-subspace embeddings, there is a line of research on the connection between eigenvalues and leverage scores, such as spectral sparsifications [45, 5].

Sublinear-time algorithms for the related graph problems has also been investigated. For example, Lee [29] gave an algorithm for producing a probabilistic $(\epsilon,\delta)$-spectral sparsifier with $O(n\log n/\epsilon^2)$ edges in $\tilde{O}(n/\epsilon^2\delta)$ time for unweighted undirected graph. Note that its running time is sublinear if the number of edges in the graph is large enough. For spectral approximations in sublinear time, various approximation guarantees have been studied in [14, 39, 7].

**Organization.** In Section 2, we provide the basic definitions and notations of this work. Next we discuss about eigenvalues of perturbed matrices and graphs in Section 3. Then we prove the lower bounds of Theorem 1 in Section 4. Finally, we discuss the approximation algorithm for degree-2 graphs in Section 5. Due to the space constraint, we leave the proof of Theorem 2 and 3 to the full version of this paper in arXiv.

## 2 Preliminaries

For any integer $k\geq 1$, let $[k]:=\{1,\cdots,k\}$. We use $a=b\pm c$ to denote $a\in[b-c,b+c]$ and $\mathbf{1}$ to denote the all 1 vector.

**Basic definitions from graph theory.** In this work, we only consider undirected graphs with unit weights on each edge. Given an undirected graph $G = (V, E)$ with $n := |V|$ vertices and $m := |E|$ edges, let $A_G \in \mathbb{R}^{n\times n}$ denote the adjacency matrix of $G$ and $D_G \in \mathbb{R}^{n\times n}$ denote its degree diagonal matrix. We use $L_G \in \mathbb{R}^{n\times n}$ to denote its Laplacian, i.e., $L_G = D_G - A_G$. Also, we use $V(G)$ and $E(G)$ to denote the vertex set and edge set of a graph $G$.

In this work, we use $\tilde{L}_G$ to denote the normalized Laplacian of $G$, i.e., $\tilde{L}_G := D_G^{-1/2} \cdot L_G \cdot D_G^{-1/2}$. When the graph is clear, we hide the notation $G$. Also, we use $V(G)$ and $E(G)$ to denote the vertex and edge set of $G$. Moreover, we use $d$ to denote the maximum degree of $G$. For a vertex $u$, let $d(u)$ denote its degree in $G$ and $\mathbf{1}_u \in \mathbb{R}^n$ denote the indicator vector of $u$, i.e., $\mathbf{1}_u(v) = 1$ if $v = u$ and 0 otherwise.

Now we define the adjacency list model for sublinear time graph algorithms [41]. There are three types of operations in constant time:

1. degree query: the algorithm queries the degree of a fixed vertex $v \in V$;
2. neighbor query: the algorithm queries the $i$-th neighbor of vertex $v$ given $v$ and $i$;
3. uniform sampling: the algorithm receives a random vertex in $V$.

**Basic definitions about matrices and expander graphs.** We use $\psi$, $\phi$, and bold letter $\mathbf{a}$ to denote vectors and $\|\cdot\|$ to denote their $L_2$ (Euclidean) norms. For a vector $\mathbf{a} \in \mathbb{R}^V$ and a subset $U \subset V$, let $\mathbf{a}(U)$ denote the vector in $\mathbb{R}^U$ which contains the corresponding entries in $U$. So $\mathbf{a}(i)$ denotes the $i$-th entry of $\mathbf{a}$.

Given a symmetric matrix $A$, we always use $\lambda_1(A) \le \lambda_2(A) \le \cdots \le \lambda_n(A)$ to denote its eigenvalues in the non-decreasing order. Furthermore, let its eigendecomposition be $A = \sum_i \lambda_i(A) \cdot \psi_i \psi_i^T$ where $\psi_i$ is the eigenvector corresponding to the eigenvalue $\lambda_i(A)$.

We say $G$ is an expander if the second smallest eigenvalue $\lambda_2(\tilde{L})$ of $\tilde{L}$ is at least $c_1$, for some universal $c_1 > 0$. This is equivalent to $\lambda_2(L_G) \ge c_2$ for some $c_2 > 0$ when the degree of $G$ is bounded. We will use Ramanujan graphs [36, 38] of degree 3 and near-Ramanujan graphs for every degree $d \ge 4$ [37]. The guarantee of a Ramanujan graph $G$ of regular degree $d$ is that $\lambda_2(L_G) \ge d - 2\sqrt{d-1}$. For near-Ramanujan graphs, we only need $\lambda_2(L_G) \ge d - 2.01\sqrt{d-1}$.

**Basic definitions about effective resistances.** Then we define the Moore-Penrose pseudo-inverse and effective resistances. For a symmetric matrix $M \in \mathbb{R}^{n\times n}$ whose eigendecomposition is $M = \sum_i \lambda_i \psi_i \psi_i^T$, its Moore-Penrose pseudo-inverse $M^\dagger = \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} \psi_i \psi_i^T$.

▶ **Definition 6** (Effective Resistances). *Given a graph $G = (V, E)$, for any two vertices $s, t \in V$, the $s - t$ effective resistance is defined as $R_G(s, t) := (\mathbf{1}_s - \mathbf{1}_t)^\top \cdot L_G^\dagger \cdot (\mathbf{1}_s - \mathbf{1}_t)$. Moreover, the total effective resistance of $G$ is defined as $R_{tot}(G) = \sum_{i<j} R_G(i, j)$.*

In this work, we will extensively use the following facts about effective resistances (see [33, 44] for their proofs).

▶ **Lemma 7.** *Given a graph $G = (V, E)$, the effective resistances in $G$ satisfy the following properties:*

1. $\sum_{(u,v)\in E} R_G(u, v) = n - 1$ *and* $R_{tot}(G) = n \cdot \sum_{i=2,\ldots,n} \frac{1}{\lambda_i(L_G)}$.
2. $2m \cdot R_G(u, v) = \kappa_G(u, v)$, *where $\kappa_G(i, j)$ is the* commute time *of a simple random walk from vertex $i$ to $j$ in $G$, i.e., the expected number of steps in a random walk starting at $i$, after vertex $j$ is visited and then vertex $i$ is reached again.*
3. $\frac{1}{2}\left(\frac{1}{d(u)} + \frac{1}{d(v)}\right) \le R_G(u, v) \le \left(1/\lambda_2(\widetilde{L}_G)\right) \cdot \left(\frac{1}{d(u)} + \frac{1}{d(v)}\right)$ *where $\lambda_2(\widetilde{L}_G)$ is the 2nd smallest eigenvalue of the normalized Laplacian $\widetilde{L}_G$.*

4. *Given any $(s,t)$, consider all functions $\phi \in \mathbb{R}^V$ such that $\phi(s)=1$ and $\phi(t)=0$, then*

$$R_G(s,t) = \frac{1}{\min\limits_{\phi\in\mathbb{R}^V:\phi(s)=1,\phi(t)=0} \sum_{(u,v)\in E}(\phi(u)-\phi(v))^2}.$$

   *In fact, the minimum value above is acheived when $\phi$ is harmonic, namely the unique solution satisfying $\phi(s)=1, \phi(t)=0$, and $\phi(v)=\frac{1}{d(v)}\sum_{(u,v)\in E}\phi(u)$ for $v\in V\setminus\{s,t\}$. An equivalent definition is $R_G(s,t)=\max\limits_{\phi\in\mathbb{R}^n:\phi\perp\mathbf{1}}\frac{\langle \mathbf{1}_s-\mathbf{1}_t,\phi\rangle^2}{\phi^\top\cdot L_G\cdot\phi}$. In particular, the leverage score $\tau_\ell$ of row $\mathbf{a}_\ell$ in $A$ also equals $\tau_\ell=\max\limits_{\phi\in\mathbb{R}^n}\frac{\langle \mathbf{a}_\ell,\phi\rangle^2}{\|A\cdot\phi\|_2^2}$.*
5. *Let $\mathcal{T}(G)$ denote all spanning trees in $G$. Let us pick $T\in\mathcal{T}$ uniformly at random. Then $R_G(u,v)$ is the probability $(u,v)$ is in $T$, i.e., $R_G(u,v)=\Pr_{T\sim\mathcal{T}(G)}[(u,v)\in T]$.*

Note that the first equation $\sum_{(u,v)\in E} R_G(u,v)=n-1$ in Lemma 7 considers the summation over all edges in $E$ where the total effective resistance is the summation over all pairs.

## 3 Eigenvalues of Perturbed Graphs and Matrices

We discuss Lemma 4 and Lemma 5 in this section. We give two different proofs for Lemma 5 such that Lemma 4 is a direct corollary by setting $A$ to be the incidence matrix. Then we show expander graphs with edges of effective resistance 1 to illustrate that we have to remove edges with $R_G(u,v)<1$ to keep the perturbed graph as an expander in Lemma 4.

We restate Lemma 5 again. One remark is that both bounds could be tight in Lemma 5. For example, consider the case $A^\top A=I$ whose $\tau_\ell=\|\mathbf{a}_\ell\|_2^2$ for all $\ell$ and $\lambda_i\equiv 1$ for all $i$. Then after removing any $\mathbf{a}_\ell$ (hence $k=1$), $(A')^\top A'=I-\mathbf{a}_\ell\cdot\mathbf{a}_\ell^\top$ has $\lambda_1'=1-\|\mathbf{a}_\ell\|_2^2$ with eigenvector $\frac{\mathbf{a}_\ell}{\|\mathbf{a}_\ell\|_2}$. So $\lambda_1'$ matches the lower bound $(1-\tau_\ell)\cdot\lambda_1$; and all the rest eigenvalues of $(A')^\top A'$ are 1, the same as those of $A^\top A$.

▶ **Lemma 5.** *Given a matrix $A\in\mathbb{R}^{m\times n}$, let $\lambda_1\le\cdots\le\lambda_n$ be the eigenvalues of $A^\top A$. Moreover, for each $\ell\in[m]$, let $\mathbf{a}_\ell$ be row $\ell$ of $A$ and $\tau_\ell=\mathbf{a}_\ell^\top(A^\top A)^\dagger\mathbf{a}_\ell$ be its leverage score.*

*For any $k$ distinct indices $\ell_1,\ldots,\ell_k\in[m]$, let $A'\in\mathbb{R}^{(m-k)\times n}$ be the matrix obtained from $A$ by removing the corresponding $k$ rows $\mathbf{a}_{\ell_1},\ldots,\mathbf{a}_{\ell_k}$; and let $\lambda_1'\le\cdots\le\lambda_n'$ be the eigenvalues of $(A')^\top A'$. It holds that*

$$\forall i\in[n], \lambda_i'\in\left[(1-\tau_{\ell_1}-\tau_{\ell_2}-\cdots-\tau_{\ell_k})\cdot\lambda_i,\lambda_i\right].$$

The first proof is based on the characteristic polynomial of $A^\top A$, which is motivated by the potential function method of classical work [5]. One ingredient in this proof is the matrix determinant lemma [5].

▶ **Lemma 8.** *If $A$ is nonsingular and $v$ is a vector, then $det(A+vv^\top)=\det(A)\cdot(1+v^\top A^{-1}v)$.*

**Proof of Lemma 5.** For ease of exposition, we start with $k=1$. Namely, we remove one row $\mathbf{a}_\ell$ from $A$ to obtain $A'$. Consider the characteristic polynomial of $(A')^\top A'$:

$$\begin{aligned} det(xI-(A')^\top A') &= det(xI-A^\top A+\mathbf{a}_\ell\mathbf{a}_\ell^\top)\\ &= det\left((xI-A^\top A)\cdot\left(I+(xI-A^\top A)^\dagger\mathbf{a}_\ell\mathbf{a}_\ell^\top\right)\right)\\ &= det\left(xI-A^\top A\right)\cdot det\left(I+(xI-A^\top A)^\dagger\mathbf{a}_\ell\mathbf{a}_\ell^\top\right). \end{aligned}$$

Let $\psi_i$ be the corresponding eigenvector of $\lambda_i$ in $A$ such that $(xI-A^\top A)^\dagger\mathbf{a}_\ell\mathbf{a}_\ell^\top=\left(\sum\limits_{i\in[n]:\lambda_i\neq x}\frac{1}{x-\lambda_i}\psi_i\psi_i^\top\right)\mathbf{a}_\ell\mathbf{a}_\ell^\top$. Then we apply the matrix determinant lemma (Lemma 8) to $det\left(I+(xI-A^\top A)^\dagger\mathbf{a}_\ell\mathbf{a}_\ell^\top\right)$ such that

$$det(xI-(A')^\top A') = det(xI-A^\top A)\cdot\left(1+\mathbf{a}_\ell^\top\cdot\sum_{i=1}^{n}\frac{1}{x-\lambda_i}\psi_i\psi_i^\top\cdot\mathbf{a}_\ell\right)$$
$$= det(xI-A^\top A)\cdot\left(1-\sum_{i=1}^{n}\frac{1}{\lambda_i-x}\langle\psi_i,\mathbf{a}_\ell\rangle^2\right).$$

Let $\lambda'_1,\lambda'_2,\cdots,\lambda'_n$ be the roots of $det(xI-(A')^\top A')$. Similar to the argument in [5]: If $\langle\psi_i,\mathbf{a}_\ell\rangle=0$, then $\lambda_i$ is a root of $det(xI-A^\top A)$, i.e., $\lambda'_i=\lambda_i$. Otherwise, $\lambda'_i\in(\lambda_{i-1},\lambda_i)$ satisfies $\sum_{j=1}^{n}\frac{1}{\lambda_j-\lambda'_i}\langle\psi,\mathbf{a}_\ell\rangle^2=1$. This is because $\lim_{x\to\lambda_{i-1}+}\sum_{i=1}^{n}\frac{1}{\lambda_i-x}\langle\psi_i,\mathbf{a}_\ell\rangle^2=-\infty$ and $\lim_{x\to\lambda_i-}\sum_{i=1}^{n}\frac{1}{\lambda_i-x}\langle\psi_i,\mathbf{a}_\ell\rangle^2=+\infty$. For the 2nd case, we show $\lambda'_i\geq(1-\tau_\ell)\lambda_i$.

Let the function $p(x):=\sum_{j=1}^{n}\frac{1}{\lambda_j-x}\langle\psi_j,\mathbf{a}_\ell\rangle^2$. If $\lambda_{i-1}\geq(1-\tau_\ell)\lambda_i$, then we have proved $\lambda'_i>(1-\tau_\ell)\lambda_i$. Otherwise we show $\lambda'_i>(1-\tau_\ell)\lambda_i$ by considering $p(x)$ in the continuous interval $\big[(1-\tau_\ell)\lambda_i,\lambda_i\big)$.

$$p\Big((1-\tau_\ell)\lambda_i\Big)=\sum_{j=1}^{n}\frac{1}{\lambda_j-(1-\tau_\ell)\lambda_i}\langle\psi,\mathbf{a}_\ell\rangle^2$$
$$\leq\sum_{j=i}^{n}\frac{1}{\lambda_j-(1-\tau_\ell)\lambda_i}\langle\psi,\mathbf{a}_\ell\rangle^2$$
(Since $\lambda_1<\cdots<\lambda_{i-1}<(1-\tau_\ell)\lambda_i$ from the assumption, their corresponding terms are negative.)
$$\leq\sum_{j=i}^{n}\frac{1}{\lambda_j-(1-\tau_\ell)\lambda_j}\langle\psi,\mathbf{a}_\ell\rangle^2=\frac{1}{\tau_\ell}\sum_{j=i}^{n}\frac{1}{\lambda_j}\langle\psi,\mathbf{a}_\ell\rangle^2$$

From the definition, $\tau_\ell=\mathbf{a}_\ell^\top(A^\top A)^\dagger\mathbf{a}_\ell=\mathbf{a}_\ell^\top\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\psi_i\psi_i^\top\right)\mathbf{a}_\ell=\sum_{i=1}^{n}\frac{1}{\lambda_i}\langle\psi_i,\mathbf{a}_\ell\rangle^2$. So $p\big((1-\tau_\ell)\lambda_i\big)\leq 1$.

On the other hand, $\lim_{x\to\lambda_i-}\sum_{i=1}^{n}\frac{1}{\lambda_i-x}\langle\psi_i,\mathbf{a}_\ell\rangle^2=+\infty$. So $p\big((1-\tau_\ell)\lambda_i\big)\leq 1$ and $p(\lambda_i-\epsilon)>1$ infer that there exists a $x\in[(1-\tau_\ell)\lambda_i,\lambda_i)$ such that $p(x)=1$, which also means that $det(xI-(A')^\top A')$ has a root $\lambda'_i\in[(1-\tau_\ell)\lambda_i,\lambda_i)$.

Next we prove $\lambda'_i\geq(1-\tau_{\ell_1}-\cdots-\tau_{\ell_k})\lambda_i$ by induction on $k$. The above calculation proves the base case of $k=1$.

For the inductive step, let $\tilde{A}$ denote the matrix after removing $\mathbf{a}_{\ell_1},\ldots,\mathbf{a}_{\ell_q}$ and $A'$ denote the matrix by removing one more edge $a_{\ell_{q+1}}$. By the induction hypothesis, $\lambda_i(\tilde{A})\in\left[\big(1-\sum_{j=1}^{q}\tau_{\ell_j}\big)\cdot\lambda_i,\lambda_i\right]$ and $(1-\sum_{j=1}^{q}\tau_{\ell_j})\cdot A^\top A\preceq\tilde{A}^\top\tilde{A}$. This implies $(\tilde{A}^\top\tilde{A})^\dagger\preceq(1-\sum_{j=1}^{q}\tau_{\ell_j})^{-1}\cdot(A^\top A)^\dagger$,

$$\tilde{\tau}_{\ell_{q+1}}=\mathbf{a}_{\ell_{q+1}}^\top(\tilde{A}^\top\tilde{A})^\dagger\mathbf{a}_{\ell_{q+1}}\leq\left(1-\sum_{j=1}^{q}\tau_{\ell_j}\right)^{-1}\mathbf{a}_{\ell_{q+1}}^\top(A^\top A)^\dagger\mathbf{a}_{\ell_{q+1}}=\left(1-\sum_{j=1}^{q}\tau_{\ell_j}\right)^{-1}\tau_{\ell_{q+1}}.$$

Using the perturbation bound for $k=1$ on $\tilde{A}$ and $A'$,

$$\lambda_i(A')\in\big[(1-\tilde{\tau}_{\ell_{q+1}})\lambda_i(\tilde{A}),\lambda_i(\tilde{A})\big]$$
$$\subseteq\left[\left(1-\left(1-\sum_{j=1}^{q}\tau_{\ell_j}\right)^{-1}\tau_{\ell_{q+1}}\right)\left(1-\sum_{j=1}^{q}\tau_{\ell_j}\right)\lambda_i,\lambda_i\right]=\left[\left(1-\sum_{j=1}^{q+1}\tau_{\ell_j}\right)\lambda_i,\lambda_i\right].\quad\blacktriangleleft$$

Next we present the 2nd proof, which is based on Property 4 of the leverage score in Lemma 7. One advantage of this proof is that it works directly on multiple edges.

**Proof of Lemma 5.** We will apply the Courant-Fischer theorem below, which shows $\lambda'_i = \min_{\substack{S\subset\mathbb{R}^n,\\ dim(S)=i}} \max_{\phi\in S} \frac{\phi^\top (A')^\top A'\phi}{\phi^\top\phi}$.

▶ **Lemma 9** (Courant-Fischer-Weyl theorem)**.** *Let $H$ be an $n\times n$ Hermitian matrix with eigenvalues $\lambda_1(H)\le\lambda_2(H)\le\cdots\le\lambda_n(H)$, then*

$$\lambda_k(H) = \min_{\substack{S\subset\mathbb{R}^n,\\ dim(S)=k}} \max_{x\in S} \frac{x^\top Hx}{x^\top x} \qquad \textit{and} \qquad \max_{\substack{S\subset\mathbb{R}^n,\\ dim(S)=n-k+1}} \min_{x\in S} \frac{x^\top Hx}{x^\top x}.$$

We compare the Rayleigh quotient $\frac{\phi^\top (A')^\top A'\phi}{\phi^\top\phi}$ with $\frac{\phi^\top A^\top A\phi}{\phi^\top\phi}$:

$$\frac{\phi^\top (A')^\top A'\phi}{\phi^\top\phi} = \frac{\|A'\phi\|_2^2}{\|\phi\|_2^2} = \frac{\|A\phi\|_2^2 - \sum_{j=1}^{k}\left(a_{\ell_j}^\top\phi\right)^2}{\|\phi\|_2^2} = \frac{\|A\phi\|_2^2}{\|\phi\|_2^2}\cdot\left[1-\sum_{j=1}^{k}\frac{\left(a_{\ell_j}^\top\phi\right)^2}{\|A\phi\|_2^2}\right].$$

Since $\tau_\ell = \max_{\phi\in\mathbb{R}^n}\frac{\left(a_\ell^\top\phi\right)^2}{\|A\phi\|_2^2}$ from Property 4 of Lemma 7, $1-\sum_{j=1}^{k}\frac{\left(a_{\ell_j}^\top\phi\right)^2}{\|A\phi\|_2^2}\ge 1-\sum_{j=1}^{k}\tau_{\ell_j}$.

Using Lemma 9 again,

$$\lambda'_i = \min_{\substack{S\subset\mathbb{R}^n,\\ dim(S)=i}}\max_{\phi\in S}\frac{\|A\phi\|_2^2}{\|\phi\|_2^2}\left[1-\sum_{j=1}^{k}\frac{\left(a_{\ell_j}^\top\phi\right)^2}{\|A\phi\|_2^2}\right] \ge \left(1-\sum_{j=1}^{k}\tau_{\ell_j}\right)\min_{\substack{S\subset\mathbb{R}^n,\\ dim(S)=i}}\max_{\phi\in S}\frac{\|A\phi\|_2^2}{\|\phi\|_2^2} = (1-\sum_{j=1}^{k}\tau_{\ell_j})\lambda_i.$$

As $1-\sum_{j=1}^{k}\frac{\left(a_{\ell_j}^\top\phi\right)^2}{\|A\phi\|_2^2}\le 1$, we can get $\lambda'_i\le\lambda_i$ in a similar way. Combining the two inequalities, $\lambda'_i\in[(1-\sum_{j=1}^{k}\tau_{\ell_j})\lambda_i,\lambda_i]$. ◀

We remark that Lemma 5 is also an instantiation of Theorem 2.8 in [26] with leverage scores. However, we believe the above two proofs shed more insights on the structure of perturbed matrices and are simplier (without Weyl's interlacing inequality). But for completeness, we provide that proof in the full version.

Next we show that there exist edges in expander graphs with unit effective resistance.

▷ Claim 10. For any $c_0>0$ and infinitely many $n$, there exists an expander graph $G$ of constant degree such that

1. The smallest non-trivial eigenvalue of its Laplacian is at least $c_0$;
2. There exists an edge $e$ in $G$ with effective resistance 1.

Hence, after removing $e$, $\lambda_2(L_{G'})=0$ in the perturbed graph $G'$ such that $G'$ is no longer an expander.

Proof. Given an expander $G'=(V',E')$ with constant degree and $\lambda_2(L_G)=\Omega(1)$, we add an extra vertex of degree 1 to it. More formally, let $V'=\{1,\ldots,n-1\}$ and $G=(V,E)=(V'\cup\{n\},E'\cup\{(n-1,n)\})$. Then $R_G(n-1,n)=1$. We will show $\lambda_2\left(L_{G'}\right)=\Omega(1)$ such that $G'$ is also an expander (its degree is still a constant). So $L_G(i,j)=L_{G'}(i,j)$ except $i\in\{n-1,n\}$ and $j\in\{n-1,n\}$ where those four entries have $L_G(n-1,n-1)=L_{G'}(n-1,n-1)+1, L_G(n-1,n)=L_G(n,n-1)=-1$, and $L_G(n,n)=1$.

By the Courant-Fischer Theorem, $\lambda_2(L_G) = \min_{\phi \perp \mathbf{1}, \|\phi\|_2 = 1} \phi^\top L_G \phi$. Then we prove $\phi^\top L_G \phi = \Omega(1)$ for all $\phi$ with $\phi \perp \mathbf{1}$ and $\|\phi\|_2 = 1$ by considering $\phi(n)$ in two cases.

If $|\phi(n)| \geq 0.8$, then $|\phi(n-1)| \leq 0.6$ because $\|\phi\|_2 = 1$. So $\phi^\top L_G \phi = \sum_{(u,v)\in E} (\phi(u) - \phi(v))^2 \geq (\phi(n) - \phi(n-1))^2 \geq 0.04$.

Otherwise $|\phi(n)| < 0.8$. For convenience, we assume $\phi(n) \in [0, 0.8)$. Let $\phi([n-1]) \in \mathbb{R}^{[n-1]}$ denotes the sub-vector on $V'$ and $\mathbf{1}([n-1])$ denotes the all 1-vector on $V'$. Then $\|\phi([n-1])\| \geq 0.6$ and

$$\phi^\top L_G \phi = \sum_{(u,v)\in E} (\phi(u) - \phi(v))^2 \geq \sum_{(u,v)\in E'} (\phi(u) - \phi(v))^2 = \phi([n-1])^\top \cdot L_{G'} \cdot \phi([n-1]).$$

Since $L_{G'}$ has an eigenvalue 0 with eigenvector $\mathbf{1}([n-1])$, we consider $\phi([n-1])$ after orthonormalization: $\phi([n-1]) - \frac{\langle \phi([n-1]), \mathbf{1}([n-1])\rangle}{n-1} \cdot \mathbf{1}([n-1])$.

Note that $\langle \phi, \mathbf{1}\rangle = 0$ implies $\langle \phi([n-1]), \mathbf{1}([n-1])\rangle = -\phi(n)$. We calculate its $L_2$ norm after orthonormalization as

$$\begin{aligned}
&\left\| \phi([n-1]) - \frac{\langle \phi([n-1]), \mathbf{1}([n-1])\rangle}{n-1} \cdot \mathbf{1}([n-1]) \right\|^2 \\
&= \left\| \phi([n-1]) + \frac{\phi(n)}{n-1} \cdot \mathbf{1}([n-1]) \right\|^2 \\
&= \|\phi([n-1])\|^2 + \left\| \frac{\phi(n)}{n-1} \cdot \mathbf{1}([n-1]) \right\|^2 + \frac{2\phi(n)}{n-1} \cdot \langle \phi([n-1]), \mathbf{1}([n-1])\rangle \\
&\geq \|\phi([n-1])\|^2 - \frac{2\phi(n)}{n-1} \cdot \phi(n) \geq 0.36 - \frac{2}{n-1} \geq 0.3.
\end{aligned}$$

So $\phi^\top L_G \phi \geq \lambda_2(L_{G'}) \cdot \left\| \phi([n-1]) - \frac{\langle \phi([n-1]), \mathbf{1}([n-1])\rangle}{n-1} \cdot \mathbf{1}([n-1]) \right\|^2 = \Omega(1)$.

From all discussion above, $\lambda_2(L_G) = \Omega(1)$. Since $G$ has a constant degree, $\lambda_2(\tilde{L}_G) \geq \lambda_2(L_G)/d(G) = \Omega(1)$, which means $G$ is also an expander. ◁

## 4 Lower Bound for Degree 3

In this section, we prove the lower bound for graphs with degrees at most 3 (except the given pair $s$ and $t$). Recall the statement of Theorem 1 in Section 1.

▶ **Theorem 1.** *There are infinitely many $n$ and graphs of $n$ vertices such that any local algorithm with success probability 0.6 and approximation ratio 1.01 on $R_G(s,t)$ needs $\Omega(n)$ queries. This holds even for graphs whose vertices are of degree at most 3 except the adjacent pair $s$ and $t$.*

One ingredient in the proof is Lemma 4, which bounds the expansion of a perturbed graph $G'$ obtained from removing one edge $e$ of the original graph $G$, in terms of the eigenvalues of the Laplacian of $G$ and the effective resistance $R_G(e)$.

In the rest of this section, we finish the proof of Theorem 1. The high level idea is to consider a graph $G$ of degree 3 (see Figure 2 for an illustration) constituted by two disjoint expanders $H_s$ and $H_t$ with one extra edge between $s$ and $t$ in these two expanders separately.

Then we produce a random graph $G'$ as follows (see Figure 1 for an illustration): remove one random edge $(u, u')$ in $H_s$ and another one $(v, v')$ in $H_t$ separately; then add two edges $(u, v)$ and $(u', v')$ to $G'$.

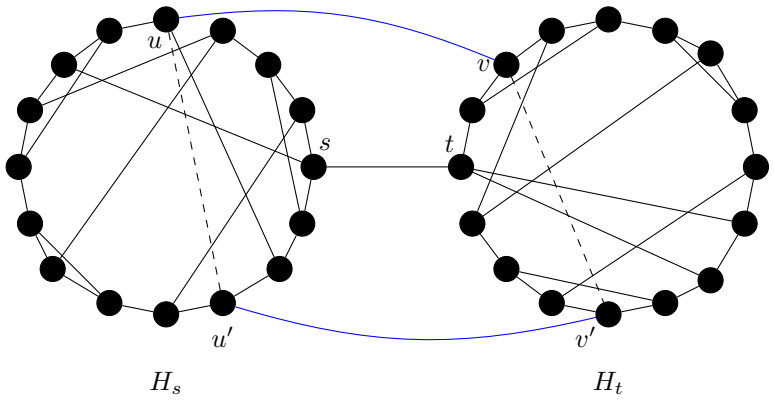



**Figure 1** $G'$ after modification.

So the number of vertices in $G$ and $G'$ are the same; moreover all vertices have the same degrees. Thus the only way to distinguish between $G$ and $G'$ is to figure out whether any of these four edges $(u, u'), (v, v'), (u, v), (u', v')$ is in the graph or not. Since there are $3n/2$ edges, this needs $\Omega(n)$ queries. Finally we bound $R_{G'}(s, t)$ (and the approximation ratio) via Lemma 4.

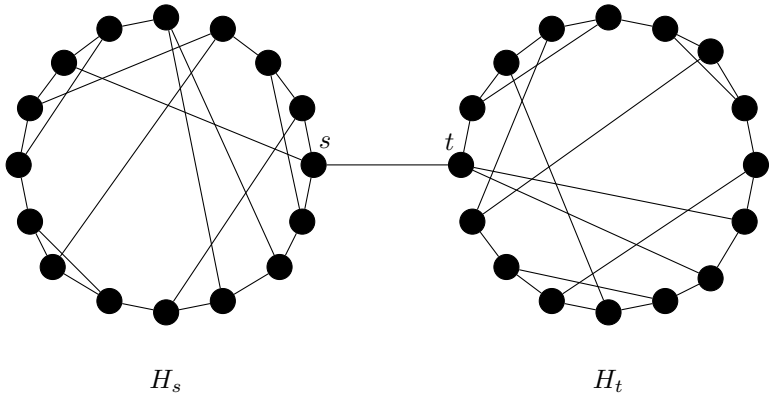



**Figure 2** Construction of $G$ with two expanders $H_s$ and $H_t$.

**Proof of Theorem 1.** We provide a distribution $\mathcal{G}$ of graphs with $n$ vertices and degree at most 3 (except $s$ and $t$). Then by Yao's minimax principle, we only need to consider deterministic algorithms of $q_n$ neighbor queries and 1.01-approximation ratio whose success probability is at least 0.6 over $\mathcal{G}$. Our goal is to prove $q_n = \Omega(n)$.

Consider any $n$ such that there exists a 3-regular Ramanujan graph $H$ of size $n/2$ [36, 38]. Then we construct $G$ with the given pair $(s, t)$ as follows:

1. Take two vertex-disjoint copies of $H$, denoted by $H_s$ and $H_t$, such that $H_s$ contains vertex $s$ and $H_t$ contains vertex $t$.
2. Define the vertex set $V(G)$ of $G$ to be the union of the vertex sets of $H_s$ and $H_t$.
3. Define the edge set $E(G)$ of $G$ to be the union of $(s, t)$, the edge set $E(H_s)$ of $H_s$ and the edge set $E(H_t)$ of $H_t$.

Note that the effective resistance $R_G(s, t) = 1$ in $G$.

Next, we construct a random graph $G'$ based on $G$. Let $E_{s,3/4}$ be the set of edges in $H_s$ with effective resistance at most 3/4, i.e., $E_{s,3/4} = \{(u,v) \in E(H_s) | R_G(u,v) \leq 3/4\}$ and we define $E_{t,3/4}$ analogously. We use the following claim to lower bound the sizes of these two sets whose proof is deferred to Section 4.1.

▷ Claim 11. It holds that $|E_{s,3/4}| \geq n/12$ and $|E_{t,3/4}| \geq n/12$.

We give the construction of $G'$ which is almost identical to $G$ except four edges.

1. Choose one edge $(u,u')$ uniformly at random from $E_{s,3/4}$ and remove it from $G$. Similarly, remove another random edge $(v,v') \in E_{t,3/4}$. For convenience, let $H'_s$ be the subgraph of $H_s$ obtained by removing $(u,u')$ from $H_s$ and $H'_t$ be the subgraph obtained by removing $(v,v')$ from $H'_t$.
2. Add $(u,v)$ and $(u',v')$ to $G'$.

By our choices of $(u,u') \in E_{s,3/4}$ and $(v,v') \in E_{t,3/4}$, $H'_s$ and $H'_t$ are still expander graphs from Lemma 4. Based on this property, we show the effective resistance between $s$ and $t$ in $G'$ is strictly less than 1 in Claim 12, whose proof is deferred to Section 4.1.

▷ Claim 12. It holds that $R_{G'}(s,t) \leq 0.99$.

Now we fix a deterministic algorithm $A$ with approximation ratio at most 1.01 and consider the underlying distribution $\mathcal{G}$, which is $G$ or $G'$ with probability 1/2 separately. Observe that whenever $A$ succeeds, $A$ is able to distinguish between $G$ and $G'$, since the ratio between $R_G(s,t) = 1$ and $R_{G'}(s,t) \leq 0.99$ is more than 1.01. For convenience, let us modify $A$ so that its output is an assertion about whether the input graph is $G$ or $G'$. By Yao's minimax principle, it holds that

$$0.6 \leq \Pr_{\mathcal{G}}[A \text{ succeeds}] = \frac{\Pr[A(G) = G]}{2} + \frac{\Pr[A(G') = G']}{2}. \tag{1}$$

Next we consider all neighbor queries made by $A$ when the underlying graph is $G$. Since $A$ and $G$ are fixed, say $A$ makes $q_n$ fixed neighbor queries on $G$. If $G'$ and $G$ provide the same answers on these neighbor queries, $A$ fails to distinguish them. But $G'$ is obtained from $G$ by removing one random edge in $E_{s,3/4}$ and another one in $E_{t,3/4}$ separately. Hence at most $q_n$ edges in $E_{s,3/4}$ will be queried; and similarly for $E_{t,3/4}$. So we bound

$$\begin{aligned}\Pr[A(G') = G'] &\leq \Pr[A(G) = G'] + \Pr[\text{One neighbor query returns different values}] \\ &\leq 1 - \Pr[A(G) = G] + q_n/|E_{s,3/4}| + q_n/|E_{t,3/4}|.\end{aligned}$$

Plugging this into (1) with the two bounds in Claim 11, we obtain $q_n \geq 0.1 \cdot \frac{n}{12}$. ◀

We remark that replacing $H$ by an expander of degree $[m/n]$ would give a lower bound $\Omega(m)$ instead of $\Omega(n)$, but this result is covered by Theorem 2 with $\ell = 1$.

## 4.1 Proofs of Claim 11 and Claim 12

We now give the proof of Claim 11.

Proof of Claim 11. Recall that $E_{s,3/4} = \{(u,v) \in E(H_s) | R_G(u,v) \leq 3/4\}$. First, for any edge $(u,v)$ in $H_s$, $R_{H_s}(u,v) = R_G(u,v)$. Since $(s,t)$ is the unique edge between $H_s$ and $H_t$, the set of spanning trees $\mathcal{T}(G)$ is generated by picking one spanning tree $T_1 \in \mathcal{T}(H_s)$ and one $T_2 \in \mathcal{T}(H_t)$ then connecting them by $\{(s,t)\}$. Then by Property 5 of Lemma 7, $R_{H_s}(u,v) = R_G(u,v)$. Thus, $E_{s,3/4} = \{(u,v) \in E(H_s) | R_{H_s}(u,v) \leq 3/4\}$.

From Property 1 of Lemma 7, $\sum_{(u,v)\in E(H_s)} R_{H_s}(u,v) = \frac{n}{2}-1$. By the definition of $E_{s,3/4}$, we have $|E(H_s)\backslash E_{s,3/4}|\cdot\frac{3}{4} \le \sum_{(u,v)\in E(H_s)\backslash E_{s,3/4}} R_{H_s}(u,v) \le \sum_{(u,v)\in E(H_s)} R_{H_s}(u,v) = \frac{n}{2}-1$, which implies that $|E(H_s)\setminus E_{s,3/4}| \le \frac{n/2-1}{3/4}$.

Since $|E(H_s)| = \frac{n}{2}\cdot\frac{3}{2}$, it holds that $|E_{s,3/4}| \ge \frac{n}{2}\cdot\frac{3}{2} - \frac{n/2-1}{3/4} \ge n/12$. ◁

While it is possible to use the Cayley graph construction of Ramanujan graphs to obtain a better bound, we did not attempt to optimize those constants in this work. Then we finish the proof of Claim 12.

Proof of Claim 12. Recall that $H'_s$ and $H'_t$ are subgraphs in $G'$ obtained by removing $(u,u')$ in $H_s$ and $(v,v')$ in $H_t$ separately. We show that the 2nd eigenvalue of $L_{H'_s}$ has $\lambda_2(L_{H'_s}) \ge \frac{3-2\sqrt{2}}{4}$. As a Ramanujan graph $H_s$, $\lambda_2(L_{H_s}) \ge 3-2\sqrt{2}$. Then we apply Lemma 4 to $\lambda_2$: since $R_{H_s}(u,u') = R_G(u,u') \le 3/4$ (from the proof of Claim 11), $\lambda_2(L_{H'_s}) \ge \lambda_2(L_{H_s})\cdot(1 - R_{H_s}(u,u')) \ge \frac{3-2\sqrt{2}}{4}$.

As removing other edges doesn't decrease the effective resistance of $(s,t)$, we ignore $(u',v')$ to give a upper bound of $R_{G'}(s,t)$. Consider another path $s-u-v-t$, we have

$$\begin{aligned} R_{H'_s}(s,u) &= (\mathbf{1}_s-\mathbf{1}_u)^\top L^\dagger_{H'_s}(\mathbf{1}_s-\mathbf{1}_u) \\ &\le \lambda_n(L^\dagger_{H'_s})\cdot\|\mathbf{1}_s-\mathbf{1}_u\|_2^2 = \frac{1}{\lambda_2(L_{H'_s})}\cdot\|\mathbf{1}_s-\mathbf{1}_u\|_2^2 = \frac{4}{3-2\sqrt{2}}\cdot 2 = 24+16\sqrt{2}; \end{aligned}$$

and the same bound holds for $R_{H'_t}(v,t)$. Since $(s,t)$ and the path passing by $s-u-v-t$ are in parallel, we have $R_{G'}(s,t) \le \frac{1}{1+\frac{1}{R_{H'_s}(s,u)+1+R_{H'_t}(v,t)}} \le 0.99$.

More formally, $1/R_{G'}(s,t) = \min_{\phi\in\mathbb{R}^V:\phi(s)=1,\phi(t)=0} \sum_{(a,b)\in E}(\phi(a)-\phi(b))^2$ from Property 4 of Lemma 7. Since $\phi(s)$ and $\phi(t)$ are fixed,

$$\min_{\phi\in\mathbb{R}^V:\phi(s)=1,\phi(t)=0} \sum_{(a,b)\in E}(\phi(a)-\phi(b))^2 = 1 + \min_{\phi\in\mathbb{R}^V:\phi(s)=1,\phi(t)=0} \sum_{(a,b)\in E(G')\backslash(s,t)}(\phi(a)-\phi(b))^2.$$

Because there is only one path $s-u-v-t$ between $s$ and $t$ in $G'$ if we ignore the two edges $(s,t)$ and $(u',v')$, we simplify the 2nd term as follows.

$$\begin{aligned} &\min_{\phi\in\mathbb{R}^V:\phi(s)=1,\phi(t)=0} \sum_{(a,b)\in E(G')\backslash(s,t)}(\phi(a)-\phi(b))^2 \\ &= \min_{\phi(u),\phi(v)\in\mathbb{R}}\left[(\phi(u)-\phi(v))^2 + \min_{\phi(s)=1}\sum_{(a,b)\in E(H'_s)}(\phi(a)-\phi(b))^2 + \min_{\phi(t)=0}\sum_{(a,b)\in E(H'_t)}(\phi(a)-\phi(b))^2\right] \\ &= \min_{\phi(u),\phi(v)\in\mathbb{R}}\left[(\phi(u)-\phi(v))^2 + \frac{(1-\phi(u))^2}{R_{H'_s}(s,u)} + \frac{\phi(v)^2}{R_{H'_t}(v,t)}\right] = \frac{1}{1+R_{H'_s}(s,u)+R_{H'_t}(v,t)}, \end{aligned}$$

where the second equation follows from Property 4 of Lemma 7 and the last equation holds when $\phi^*(u) = \frac{1+R_{H'_t}(v,t)}{1+R_{H'_s}(s,u)+R_{H'_t}(v,t)}$ and $\phi^*(v) = \frac{R_{H'_t}(v,t)}{1+R_{H'_s}(s,u)+R_{H'_t}(v,t)}$. ◁

## 5 Approximation Algorithm for Degree 2

In Section 4, we proved a lower bound for degree-3 graphs (except $s$ and $t$). Now we give an approximation algorithm to solve the case of degree 2 in sublinear time. Since all vertices are of degree 2 except $s$ and $t$, essentially, $G$ is constituted by several disjoint paths (some of them are disconnected) between $s$ and $t$.

▶ **Theorem 13.** *Given any $\epsilon < 0.1$, for any graph with degree at most 2 except the adjacent pair $s$ and $t$, there is a local algorithm such that with probability 0.99, it outputs a $(1+\epsilon)$-approximation of $R_G(s,t)$ in time $O\left(\min\{d(s)^2, d(t)^2\} \cdot \log^2 n \cdot \log\log n \cdot \epsilon^{-2}\right)$.*

Note that as long as $\min\{d(s), d(t)\}$ is not too large, the running time in the above theorem is sublinear for any constant $\epsilon > 0$. In the rest of this section, we finish the proof of Theorem 13. In fact, our algorithm could output an additive-error approximation $\tilde{R}_G(s,t)$ in time $O\left(\min\{d(s), d(t)\} \cdot \log^2 n \cdot \log\log n \cdot \delta^{-1} \cdot \epsilon^{-1}\right)$ such that $\tilde{R}_G(s,t) = (1\pm\epsilon)\cdot R_G(s,t) + \delta$. Since $R_G(s,t) \geq \frac{2}{1/d(s)+1/d(t)} \geq 1/\min\{d(s), d(t)\}$ from Property 3 of Lemma 7, $\tilde{R}_G(s,t)$ is a $(1+2\epsilon)$-approximation if we choose $\delta = \epsilon/\min\{d(s), d(t)\}$. This choice of $\delta$ gives the running time in Theorem 13. Then we show the additive-error approximation.

First of all, by the following claim (whose proof is deferred to the full version), we focus on the approximation of $\rho := 1/R_G(s,t)$ instead of the approximation of $R_G(s,t)$.

▷ Claim 14. If $\tilde{\rho} = (1\pm\epsilon)\cdot\rho \pm \delta$ for $\epsilon < 0.1$, $\delta < 0.1$, and $\rho \geq 1$, we always have that $1/\tilde{\rho}$ is a $(1+2\epsilon, 2\delta)$-approximation of $1/\rho$.

We remark that $\rho \geq 1$ for adjacent pairs. But for any non-adjacent pair $s$ and $t$, our algorithm gives an additive-error approximation on their conductance $1/R_G(s,t)$.

Because all vertices except $s$ and $t$ are of degree 2, $G$ is constituted by several *disjoint* paths (some are disconnected) between $s$ and $t$. For convenience, we assume $d(s) \leq d(t)$ and consider each edge from $s$ as a path. In this proof, we define its length $\ell_i = +\infty$ if it is disconnected; otherwise $\ell_i$ is the exact length of that path from $s$ to $t$. Since $R_G(s,t) = \frac{1}{\sum_i 1/\ell_i}$, our goal becomes to approximate $1/R_G(s,t) = \rho = \sum_i 1/\ell_i$. In Algorithm 1, we describe the additive-error approximation algorithm motivated by the chaining argument whose idea is to sample longer paths with smaller probability. Its guarantee is

$$\tilde{\rho} = (1\pm\epsilon)\cdot\rho \pm \delta \qquad \text{for the output } \tilde{\rho} = 1/\tilde{R}_G(s,t) \text{ and } \rho = 1/R_G(s,t), \tag{2}$$

which gives the additive error approximation $\tilde{R}_G(s,t) = (1\pm 2\epsilon)\cdot R_G(s,t) \pm 2\delta$ by Claim 14.

**Algorithm 1** Algorithm for estimating effective resistance with additive error.

**function** AdditiveApproximationEffectiveResistance($G, \epsilon, \delta, s, t$)
  If $d(s) > d(t)$ then swap $s$ and $t$
  $\tilde{\rho} \leftarrow 0, p_0 \leftarrow 1$
  $a \leftarrow \dfrac{20\log n \cdot \log\log n}{\epsilon\delta}$
  **for** $k$ **in** $[0, \cdots, \log n]$ **do**
    **for** each path $i$ from $s$ **do**
      With prob. $p_k$, take at most $2^{k+1}\cdot a$ steps in path $i$ to find out its length $\ell_i$
      if it reaches $t$ and $\ell_i > 2^k \cdot a$, then $\tilde{\rho} \leftarrow \tilde{\rho} + 1/(p_k \cdot \ell_i)$
    $p_k \leftarrow p_{k-1}/2$
  **return** $1/\tilde{\rho}$ as the effective resistance (and $\tilde{\rho}$ as the conductance)

In the rest of this section, we prove the correctness of Algorithm 1. We remark that $p_k = 2^{-k}$ in this section. Let us consider its expected time. For a path of length $\ell_i$, if $j = \lfloor \log_2 \ell_i/a \rfloor$, the number of expected steps on this path is at most

$$p_0\cdot 2a + p_1\cdot 4a + \cdots + p_j\cdot 2^j a + p_{j+1}\cdot\ell_i + \cdots + p_{\log n}\cdot\ell_i = 2a\cdot j + \ell_i\cdot(p_{j+1}+\cdots+p_{\log n}) = O(a\cdot\log n).$$

Since we can start from either $s$ or $t$, the expected time is $O(\min\{d(s), d(t)\}\cdot a\cdot\log n)$.

Next we show the approximation guarantee in (2). For all paths whose lengths are at most $2a$, the algorithm gets the exact conductance $1/\ell_i$ of them. The rest paths are divided into $\log n$ groups $G_1, \ldots, G_{\log n}$. The paths in group $G_k$ have lengths between $(2^k \cdot a, 2^{k+1} \cdot a]$. Assuming path $i$ is in group $k$, we define the random variable $X_i$ for estimating $1/\ell_i$: With probability $p_k$, $X_i = \frac{1}{p_k \ell_i} - \frac{1}{\ell_i}$; otherwise, $X_i = -\frac{1}{\ell_i}$. So $\mathbb{E}\, X_i = 0$ and $\mathsf{Var} X_i = \frac{1-p_k}{p_k \cdot \ell_i^2} \le \frac{1}{p_k \cdot \ell_i^2}$. And the result of Algorithm 1 is $\tilde{\rho} = \sum \frac{1}{\ell_i} + X_i = \rho + \sum X_i$.

Now, we calculate the approximation of group $G_k$ by Bernstein's inequality. There are two cases of $G_k$, i.e., $\sum_{G_k} \frac{1}{\ell_i} > b$ and $\sum_{G_k} \frac{1}{\ell_i} \le b$ for a threshold value $b = \frac{\delta}{\epsilon \cdot \log n}$.

▶ **Lemma 15** (Bernstein's inequality [6]). *Let $X_1, \cdots, X_n$ be independent zero-mean random variables. Suppose that $|X_i| \le M$ almost surely, for all $i$. Then, for all positive $t$,* $\Pr\left[\left|\sum_{i=1}^{n} X_i\right| \ge t\right] \le 2\exp\left[-\frac{1}{4} \cdot \min\left\{t^2 \Big/ \sum_{i=1}^{n} \mathbb{E}\, X_i^2, t/M\right\}\right]$.

In the previous case, $(\sum_{G_k} \frac{1}{\ell_i})^2 / \sum_{G_k} \frac{1}{\ell_i^2} \ge \frac{(|G_k| \cdot \min_{G_k} \ell_i)^2}{|G_k| \cdot (\max_{G_k} \ell_i)^2} \ge |G_k|/4 \ge 2^{k-2} \cdot ab$. The first inequality comes from $\max_{G_k} \ell_i < 2 \min_{G_k} \ell_i$. And the last step is implied by $b < \sum_{G_k} \frac{1}{\ell_i} < |G_k|/(2^k \cdot a)$ since each $\ell_i > 2^k \cdot a$ in $G_k$.

Then $\sum_{G_k} \mathbb{E}\, X_i^2 \le \sum_{G_k} \frac{1}{p_k \ell_i^2} \le \frac{4}{ab} (\sum_{G_k} \frac{1}{\ell_i})^2$, and $|X_i| \le \frac{2^k - 1}{2^k \cdot a} \le \frac{1}{a}$ for all $i \in G_k$. From the above discussion, Bernstein's inequality infers that for $t = \epsilon \sum_{G_k} \frac{1}{\ell_i}$,

$$\Pr\left[\left|\sum_{G_k} X_i\right| \ge \epsilon \sum_{G_k} \frac{1}{\ell_i}\right] \le 2\exp\left[-\frac{1}{4} \min\left\{\frac{ab\epsilon^2}{4}, a\epsilon \sum_{G_k} \frac{1}{\ell_i}\right\}\right] \le 2\exp\left[-\frac{1}{4} \min\left\{\frac{ab\epsilon^2}{4}, ab\epsilon\right\}\right].$$

The second step is by the assumption $\sum_{G_k} \frac{1}{\ell_i} > b$.

In the latter case, $\sum_{G_k} \mathbb{E}\, X_i^2 \le \sum_{G_k} \frac{1}{p_k \ell_i^2} \le \sum_{G_k} \frac{1}{a \cdot \ell_i} \le \frac{b}{a}$ with $\ell_i \ge 2^k \cdot a$ and $\sum_{G_k} \frac{1}{\ell_i} \le b$. So

$$\Pr\left[\left|\sum_{G_k} X_i\right| \ge \frac{\delta}{\log n}\right] \le 2\exp\left[-\frac{1}{4} \min\left\{\frac{a\delta^2}{b \log^2 n}, \frac{a\delta}{\log n}\right\}\right].$$

Take a union bound for all $G_k$ with $b = \frac{\delta}{\epsilon \cdot \log n}$, $\Pr\left[\left|\sum_{i \in [d]} X_i\right| \ge \epsilon \cdot \sum_{i \in [d]} \frac{1}{\ell_i} + \delta\right]$ is at most $2\log n \cdot \exp\left[-\frac{1}{4} \min\left\{\frac{a\delta\epsilon}{4 \log n}, \frac{a\delta}{\log n}\right\}\right] + 2\log n \cdot \exp\left[-\frac{1}{4} \min\left\{\frac{a\delta\epsilon}{\log n}, \frac{a\delta}{\log n}\right\}\right] \le 4 \log n \cdot \exp\left[-\frac{a\delta\epsilon}{16 \log n}\right]$. When $a = \frac{20 \log n \cdot \log\log n}{\epsilon\delta}$, $\tilde{\rho}$ is a $(1+\epsilon, \delta)$-approximation of $\rho$ with probability 0.99.

## References


1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Proc. of the 57th FOCS*, pages 335–344, 2016.

2 Tasweer Ahmad, Lianwen Jin, Luojun Lin, and GuoZhi Tang. Skeleton-based action recognition using sparse spatio-temporal gcn with edge effective resistance. *Neurocomputing*, 423:389–398, 2021.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

4 Alexandr Andoni, Robert Krauthgamer, and Yosef Pogrow. On solving linear systems in sublinear time. In *10th Innovations in Theoretical Computer Science Conference (ITCS)*, 2018.

5 Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM J. Comput.*, 41(6):1704–1721, 2012. doi:10.1137/090772873.

6 Serge Bernstein. *Sur l'ordre de la meilleure approximation des fonctions continues par des polynômes de degré donné*, volume 4. Hayez, imprimeur des académies royales, 1912.

7 Vladimir Braverman, Aditya Krishnan, and Christopher Musco. Sublinear time spectral density estimation. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1144–1157. ACM, 2022. doi:10.1145/3519935.3520009.

8 James R Bunch and John E Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974.

9 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1135–1146. IEEE, 2020.

10 Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proc. of the 43rd STOC*, pages 273–282, 2011.

11 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 361–372. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00042.

12 Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference, STOC'13*, pages 81–90. ACM, 2013. doi:10.1145/2488608.2488620.

13 Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m\log^{1/2}n$ time. In *Proc. of the 46th STOC*, pages 343–352, 2014.

14 David Cohen-Steiner, Weihao Kong, Christian Sohler, and Gregory Valiant. Approximating the spectrum of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018*, pages 1263–1271. ACM, 2018. doi:10.1145/3219819.3220119.

15 Michael Dinitz, Robert Krauthgamer, and Tal Wagner. Towards resistance sparsifiers. In *Proc. of the 18th APPROX*, pages 738–755, 2015.

16 Petros Drineas and Michael W. Mahoney. Effective resistances, statistical leverage, and applications to linear equation solving. Technical report, available at arXiv:1005.3097, 2010.

17 David Durfee, Yu Gao, Gramoz Goranci, and Richard Peng. Fully dynamic effective resistances. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 914–925. ACM, 2019.

18 David Durfee, Rasmus Kyng, John Peebles, Anup B Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 730–742. ACM, 2017.

19 Wendy Ellens, FM Spieksma, P Van Mieghem, A Jamakovic, and RE Kooij. Effective graph resistance. *Linear algebra and its applications*, 435(10):2491–2506, 2011.

20 Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM review*, 50(1):37–66, 2008.

21 Gramoz Goranci, Monika Henzinger, and Pan Peng. The power of vertex sparsifiers in dynamic graph algorithms. In *Proc. of the 25th ESA*, volume 87, pages 45:1–45:14, 2017.

**22** Gramoz Goranci, Monika Henzinger, and Pan Peng. Dynamic effective resistances and approximate schur complement on separable graphs. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

**23** John T Holodnak, Ilse CF Ipsen, and Thomas Wentworth. Conditioning of leverage scores and computation by qr decomposition. *SIAM Journal on Matrix Analysis and Applications*, 36(3):1143–1163, 2015.

**24** Oscar H Ibarra, Shlomo Moran, and Roger Hui. A generalization of the fast lup matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, 1982.

**25** Ilse Ipsen and Thomas Wentworth. Sensitivity of leverage scores and coherence for randomized matrix algorithms. In *Workshop on Advances in Matrix Functions and Matrix Equations, Manchester, UK*, 2013.

**26** Ilse C. F. Ipsen. Relative perturbation results for matrix eigenvalues and singular values. *Acta Numerica*, 7:151–201, 1998. doi:10.1017/S0962492900002828.

**27** Arun Jambulapati and Aaron Sidford. Efficient $\tilde{O}(n/\varepsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503. SIAM, 2018.

**28** Jonathan A Kelner, Gary L Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1–18. ACM, 2012.

**29** Yin Tat Lee. Probabilistic spectral sparsification in sublinear time. *arXiv preprint arXiv:1401.0085*, 2013.

**30** Huan Li and Zhongzhi Zhang. Kirchhoff index as a measure of edge centrality in weighted networks: Nearly linear time algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2377–2396. SIAM, 2018.

**31** Lawrence Li and Sushant Sachdeva. A new approach to estimating effective resistances and counting spanning trees in expander graphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2728–2745. SIAM, 2023.

**32** Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, 1979.

**33** László Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46, 1993.

**34** Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *Proc. of the 57th FOCS*, pages 593–602, 2016.

**35** Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 2019–2036. Society for Industrial and Applied Mathematics, 2015.

**36** Grigorii Aleksandrovich Margulis. Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problemy peredachi informatsii*, 24(1):51–60, 1988.

**37** Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-ramanujan graphs of every degree. *SIAM Journal on Computing*, 51(3):1–23, 2022. doi:10.1137/20M1342112.

**38** M. Morgenstern. Existence and explicit constructions of q + 1 regular ramanujan graphs for every prime power q. *Journal of Combinatorial Theory, Series B*, 62(1):44–62, 1994.

**39** Cameron Musco, Praneeth Netrapalli, Aaron Sidford, Shashanka Ubaru, and David P. Woodruff. Spectrum approximation beyond fast matrix multiplication: Algorithms and hardness. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018*, volume 94 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.8.

**40** Pan Peng, Daniel Lopatta, Yuichi Yoshida, and Gramoz Goranci. Local algorithms for estimating effective resistance. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1329–1338, 2021.

41 Dana Ron. Sublinear-time algorithms for approximating graph parameters. In *Computing and Software Science*, pages 105–122. Springer, 2019.

42 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 509–517. IEEE Computer Society, 2004.

43 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. *Proc. of the 50th STOC*, 2018.

44 Daniel A. Spielman. Spectral and algebraic graph theory. Draft of textbook, available at `http://cs-www.cs.yale.edu/homes/spielman/sagt/`, 2019.

45 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011.

46 Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Analysis Applications*, 35(3):835–885, 2014.

47 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10(1-2):1–157, 2014. `doi:10.1561/0400000060`.

# New Menger-Like Dualities in Digraphs and Applications to Half-Integral Linkages

**Victor Campos**
ParGO group, Universidade Federal do Ceará, Fortaleza, Brazil

**Jonas Costa**
ParGO group, Universidade Federal do Ceará, Fortaleza, Brazil

**Raul Lopes**
DIENS, École normale supérieure de Paris, CNRS, France
Université Paris-Dauphine, PSL University, CNRS UMR7243, Paris, France

**Ignasi Sau**
LIRMM, Université de Montpellier, CNRS, Montpellier, France

## Abstract

We present new min-max relations in digraphs between the number of paths satisfying certain conditions and the order of the corresponding cuts. We define these objects in order to capture, in the context of solving the half-integral linkage problem, the essential properties needed for reaching a large bramble of congestion two (or any other constant) from the terminal set. This strategy has been used ad-hoc in several articles, usually with lengthy technical proofs, and our objective is to abstract it to make it applicable in a simpler and unified way. We provide two proofs of the min-max relations, one consisting in applying Menger's Theorem on appropriately defined auxiliary digraphs, and an alternative simpler one using matroids, however with worse polynomial running time.

As an application, we manage to simplify and improve several results of Edwards et al. [ESA 2017] and of Giannopoulou et al. [SODA 2022] about finding half-integral linkages in digraphs. Concerning the former, besides being simpler, our proof provides an almost optimal bound on the strong connectivity of a digraph for it to be half-integrally feasible under the presence of a large bramble of congestion two (or equivalently, if the directed tree-width is large, which is the hard case). Concerning the latter, our proof uses brambles as rerouting objects instead of cylindrical grids, hence yielding much better bounds and being somehow independent of a particular topology.

We hope that our min-max relations will find further applications as, in our opinion, they are simple, robust, and versatile to be easily applicable to different types of routing problems in digraphs.

**2012 ACM Subject Classification** Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

**Keywords and phrases** directed graphs, min-max relation, half-integral linkage, directed disjoint paths, bramble, parameterized complexity, matroids

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.30

**Related Version** *Full Version*: https://arxiv.org/abs/2306.16134

**Funding** *Victor Campos*: FUNCAP/PRONEM PNE-0112-00061.01.00/16, and CAPES/Cofecub 88887.712023/2022-00.
*Jonas Costa*: FUNCAP/PRONEM PNE-0112-00061.01.00/16, FUNCAP PS1-0186-00155.01.00/2, and PhD scholarship granted by CAPES.
*Raul Lopes*: group Casino/ENS Chair on Algorithmics and Machine Learning, and French National Research Agency under JCJC program (ASSK: ANR-18-CE40-0025-01).
*Ignasi Sau*: ELIT (ANR-20-CE48-0008-01).




## 1 Introduction

In combinatorial optimization, a *min-max relation* establishes the equality between two quantities, one naturally associated with *minimizing* the size of an object satisfying some conditions, and the other one associated with *maximizing* the size of another object. Within graph theory, famous such min-max relations include Kőnig's Theorem [15] stating the equality between the sizes of a maximum matching and a minimum vertex cover in a bipartite graph or, more relevant to this article, Menger's Theorem [18] stating, in its simplest form, the equality between the maximum number of pairwise internally disjoint paths between two vertices, and the minimum size of a vertex set disconnecting them. Typically, min-max relations come along with polynomial-time algorithms to find the corresponding objects, making them extremely useful from the algorithmic point of view.

In this article we focus on directed graphs, or *digraphs* for short, and our results are motivated by the complexity of problems related to finding *directed disjoint paths* between given terminals. More precisely, in the $k$-Directed Disjoint Paths problem ($k$-DDP for short), we are given a digraph $G$ and $k$ pairs of vertices $s_i, t_i, i \in [k]$, and the objective is to decide whether $G$ contains $k$ pairwise disjoint paths connecting $s_i$ to $t_i$ for $i \in [k]$. A solution to this problem is usually called a *linkage* in the literature. Here we note that disjoint paths is equivalent to paths which are vertex-disjoint.

Unfortunately, Fortune et al. [9] proved that the $k$-DDP problem is NP-complete already for $k = 2$, and Thomassen [20] strengthened this result by showing that it remains so even if the input digraph is $p$-strongly connected (see Section 2 for the definition) for any integer $p \geq 1$. Thus, in order to obtain positive algorithmic results, research has focused on either restricting the input digraphs (for instance, to being acyclic [19] or, more generally, to having bounded directed tree-width [12]), or on considering relaxations of the problem. Concerning the latter, a natural candidate is to relax the disjointness condition of the paths, and allow for *congestion* in the vertices. Namely, for an integer $c \geq 2$, an input of the $k$-Directed $c$-Congested Disjoint Paths problem ($(k, c)$-DDP for short) is the same as in the $k$-DDP, but now we allow each vertex of $G$ to occur in at most $c$ out of the $k$ paths connecting the terminals. In the particular case $c = 2$, a solution to this problem is usually called a *half-integral linkage* in the literature.

Despite a considerable number of attempts, it is still open whether the $(k, c)$-DDP problem can be solved in polynomial time for every fixed value of $c \geq 2$ and $k > c$ (note that if $k \leq c$, then the problem can be easily solvable in polynomial time just by verifying the connectivity between each pair of terminals). A positive answer for the case $c = 2$ has been recently conjectured by Giannopoulou et al. [11]. Again, in order to obtain positive results, several restrictions and variations of the problem have been considered, such as considering several parameterizations [2, 16], restricting the input graph to have high connectivity [8], or considering an *asymmetric* version of the $(k, c)$-DDP problem [11, 13, 14], where the input is as in $(k, c)$-DDP, but the goal is to either certify that it is a no-instance of $k$-DDP (without congestion) or a yes-instance of $(k, c)$-DDP. This asymmetric version has been solved in polynomial time for every fixed $k$ (i.e., showing that it is in XP; see Section 2) for distinct values of $c$ in a series of articles, namely for $c = 4$ by Kawarabayashi et al. [13], for $c = 3$ by Kawarabayashi and Kreutzer [14], and for $c = 2$ by Giannopoulou et al. [11].

The main motivation of this article stems from the techniques used in the latter two approaches mentioned above. In a nutshell, the main strategy used in [8, 11, 13, 14] is the following. First, one computes whether the directed tree-width of the input graph is bounded by an appropriate function of $k$, the number of terminal pairs. This can be done in time XP

in $k$ by the results of Johnson et al. [12], or even in time FPT by the results of Campos et al. [4] (see also [3, Theorem 9.4.4]). If the directed tree-width is bounded by a function of $k$, one solves the problem in time XP by using standard programming techniques from Johnson et al. [12] (cf. Proposition 3). If not, one exploits the fact that large directed tree-width implies the existence of large "structures" that can be used to carry out the routing of the desired paths. Typically, such a structure is a *bramble*, as for example in [8], or a *cylindrical grid*, as for example in [11, 14], making use of the celebrated Directed Grid Theorem of Kawarabayashi and Kreutzer [14] (see also [4] for recent improvements). For the sake of exposition, assume henceforth that the desired structure is a bramble, but the strategy is essentially the same with a cylindrical grid.

A bramble in a digraph $D$ is a set $\mathcal{B}$ of strongly connected subgraphs of $G$ that pairwise either intersect or have edges in both directions. The *order* of a bramble $\mathcal{B}$ is the smallest size of a vertex set of $G$ that intersects all its elements, and its *congestion* is the maximum number of times that a vertex of $G$ appears in the elements of $\mathcal{B}$. It is known that large directed treewidth implies the existence of a bramble of large order and of congestion $c \geq 2$. For $c = 2$, a proof about how to find such a bramble in polynomial time (with degree not depending on $k$), provided that a certificate for large directed tree-width is given, can be found in [8] (see also [17] for improved bounds for brambles of higher congestion). Assume for simplicity that $c = 2$, let $\mathcal{B}$ be such a bramble, and let $S$ and $T$ be the sets of sources and sinks, respectively, of the corresponding problem. The idea is that if one can find a set $\mathscr{P}^S$ of disjoint paths from $S$ to appropriate elements of $\mathcal{B}$, and a set $\mathscr{P}^T$ of disjoint paths from appropriate elements of $\mathcal{B}$ to $T$ (regardless of the ordering of the vertices of $S$ and $T$), then we are done. Indeed, once the paths starting in $S$ reach $\mathcal{B}$, one can use the connectivity properties of the bramble to "shuffle" the paths appropriately as required by the terminals, and then follow the paths from $\mathcal{B}$ to $T$. The fact that the bramble has congestion two, and that the paths in $\mathscr{P}^S$, as well as those in $\mathscr{P}^T$, are pairwise disjoint, together with a good choice for the destination and starting points of the those paths, implies that every vertex of $G$ occurs in at most two of the resulting paths.

Otherwise, if such sets of paths $\mathscr{P}^S$ and $\mathscr{P}^T$ do *not* exist, the approach consists in using a Menger-like min-max duality to obtain an appropriate *separator* (or *cut*) between the terminals and the bramble of size bounded by a function of $c$ and $k$, and make some progress toward the resolution of the problem, for instance by splitting into subproblems of lower complexity. The ways to define and to exploit such a separator depend on every particular application, and this ad-hoc subroutine is usually one of the most technically involved parts of the resulting algorithms [8, 11, 13, 14].

**Our results and techniques.** Motivated by the inherent common essential strategy in the above articles, we aim at finding the crucial general ingredient that can be applied in order to define and find the corresponding separators. To this end, we introduce new objects that abstract the existence of the aforementioned desired paths $\mathscr{P}^S$ and $\mathscr{P}^T$ between the terminals and the bramble. These objects are what we call *D-paths*, *T-paths* and *R-paths*. The inspiration for D-paths and T-paths is what we believe to be the common essential strategies used by Edwards et al. [8] and Giannopoulou et al. [10]. We remark that a particular set of T-paths is directly constructed in [10], particularly inside the proof of [10, Theorem 9.1]. The presence of D-paths and T-paths in [8] is more subtle in the construction of a long algorithm and a collection of non-trivial proofs. In fact, they are not explicitly built due to constraints in their techniques, but our initial results for this paper included simplifying and improving the proofs in [8] using D-paths and T-paths. Once the new proofs were obtained,

we noticed that they could be further improved by a new object which we call R-paths. It must be noted that this paper includes results for D-paths, T-paths, and R-paths but we only show applications for R-paths. The reason for this is twofold. First, T-paths have stronger properties than R-paths which might be useful for solving different problems, and D-paths are needed in the proof of the duality of T-paths. And second, these three objects are similar to the point of having similar proofs for their min-max formulas and algorithms, and showing these variations incurs little additional effort. The formal definitions of these special types of paths can be found in Section 3.

All three types of paths are associated with a defining partition $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ sharing some properties and differing in others. For all three, it holds that any two paths in a same part $\mathcal{P}_i$ are disjoint, and it is possible that two paths in distinct parts share vertices. In the context of the $(k, c)$-DDP problem, the main difference between D-paths, T-paths, and R-paths lies in how we want to reach (or, by applying a simple trick of reversing the edges of the digraph, be reached from) the elements of a bramble $\mathcal{B}$ in the given digraph. In D-paths we ask that all paths end in distinct vertices of a given $B \subseteq V(G)$. In T-paths we ask that all paths end in distinct vertices of the bramble, and that the set containing all last vertices of the T-paths forms a *partial transversal* (see Section 2 for the definition) of $\mathcal{B}$. In R-paths we ask that all R-paths end in elements of $\mathcal{B}$ and that there is a "matching-like" association between the last vertices of the paths and elements of $\mathcal{B}$ containing these vertices.

For each type of paths we define an associated notion of *cut* and its corresponding *order* (see e.g. Definition 10). These cuts are respectively called *D-cuts*, *T-cuts*, and *R-cuts*. We show that each of these special types of paths and cuts satisfies a Menger-like min-max duality, that is, that the maximum number of paths equals the minimum order of a cut (cf. Theorems 11, 13, and 15). Moreover, the corresponding objects attaining the equality can be found in polynomial time. The proofs of these min-max relations basically consist in applying Menger's Theorem [18] in appropriately defined auxiliary graphs. We provide alternative simpler proofs of these equalities using intersections and unions of matroids, namely gammoids and transversal matroids. Even if the resulting polynomial-time algorithms using matroids have worse running time than the ones that we obtain by applying Menger's Theorem [18], and that using the deep theory of matroids somehow sheds less light on interpreting the actual behavior of the considered objects, we think that it is interesting to observe that the paths and cuts that we define are in fact matroids with nice properties.

The main application we provide for R-paths/cuts in this article is in the proof of Theorem 16, which is an improved version of [10, Theorem 9.1] both in the requested order of the structure and by relying on brambles instead of cylindrical grids. Informally, Theorem 16 says that given a digraph $G$, ordered sets $S, T \subseteq V(G)$, and a large (depending on the congestion $c$ and on the size $k$ of $S$ and $T$) bramble of congestion $c$, we can either find a large set of R-paths from $S$ to the bramble and from the bramble to $T$, which in turn are used to appropriately connect the pairs $s_i \in S, t_i \in T$, or find a separator of size at most $k-1$ intersecting every path from $S$ to a large subset of the bramble, or every path from a large subset of the bramble to $T$. Additionally, if the bramble is given, one of the outputs can be obtained in polynomial time, computing either the paths or one of the separators.

Since in $k$-strong digraphs the separators are never found, Theorem 16 immediately implies an improved version of a result by Edwards et al. [8]. Namely, in [8, Theorem 11] the authors show that, when restricted to $(36k^3 + 2k)$-strong digraphs, every instance of $(k, 2)$-DDP where the input digraph contains a large (depending on $k$) bramble of congestion two is positive and a solution can be found in polynomial time. When compared to theirs, our result is an improvement in the following ways. First, it allows us to solve $(k, c)$-DDP for any

$c \geq 2$ in the larger class of $k$-strong digraphs instead of being restricted to $(36k^3 + 2k)$-strong digraphs as in [8]. This bound on the strong connectivity of the digraph is almost best possible according to [8, Theorem 2], unless $\mathsf{P} = \mathsf{NP}$ (note that an $\mathsf{XP}$ algorithm in $(k-d)$-strong digraphs, for some constant $d$, may still be possible). Second, we show how to find the desired paths using a bramble of congestion $c$ and size at least $2k(c \cdot k - c + 2) + c(k-1)$, which is equal to $4k^2 + 2k - 2$ when $c = 2$, instead of the size $188k^3$ required in [8]. Finally, our proof is much simpler and shorter than the proof presented in [8]. A main reason of this simplification is that we can replace the seven properties of the paths requested in [8, Lemma 12] by $\mathsf{R}$-paths. It is worth mentioning that our algorithm reuses the procedure of Edwards et al. [8] to find a large bramble of congestion two in digraphs of large directed tree-width (cf. Corollary 8).

We remark that it is also possible to improve the result by Edwards et al. [8] from $(36k^3 + 2k)$-strong digraphs to $k$-strong digraphs by replacing part of their proof, namely [8, Theorem 11], by the result of Giannopoulou et al. [10, Theorem 9.1]. The trade-off is that the latter relies on the stronger structure of a cylindrical grid (and such grids do contain brambles of congestion two [8]) instead of brambles. A fundamental difference stands on the fact that, given a certificate of large directed tree-width, one can produce a bramble of congestion two in polynomial time, while finding a cylindrical grid still requires $\mathsf{FPT}$ time parameterized by the order of the certificate [4]. Our result using $\mathsf{R}$-paths keeps the best of both worlds: we are able to drop the request on the strong connectivity of the digraph to $k$ while relying only on brambles as the routing structures.

Our second application deals with the asymmetric version of the $(k, c)$-DDP problem discussed in the introduction. By using our min-max relations, we manage to simplify and improve one of the main results of Giannopoulou et al. [11] for the case $c = 2$. Instead of using the Directed Grid Theorem [14] to reroute the paths through a cylindrical grid, we reroute them through a bramble of congestion two in a very easy manner after a careful choice of the paths reaching and leaving the bramble, which is done by applying the duality between $\mathsf{R}$-paths and $\mathsf{R}$-cuts. Namely, we can replace [10, Theorem 9.1] (this is the full version of [11]) entirely by Theorem 16 and mostly keep the remaining part of their proof to obtain an improved version of their $\mathsf{XP}$ algorithm for the asymmetric version of $(k, 2)$-DDP.

We hope that our min-max relations will find further applications in the future as, in our opinion, they are quite simple, robust, and versatile to be easily applicable to different types of routing problems in digraphs. A natural candidate is the $(k, c)$-DDP problem for any choice of fixed values of $c \geq 2$ and $k > c$, which has remained elusive for some time.

**Organization.** In Section 2 we present some preliminaries. In Section 3 we state and discuss our new Menger-like statements for paths in digraphs. The applications of our results are presented in Section 4. Due to space limitations, the proofs of the results marked with "$(\star)$" can be found in the full version of this paper, available at `https://arxiv.org/abs/2306.16134`.

## 2 Preliminaries

Due to space limitations, in this section we provide only the most important or non-standard preliminaries, and additional ones can be found in the full version of this paper, namely basic definitions of digraphs and matroids.

We refer the reader to [5, 7] for background on parameterized complexity, and we define here only the most basic definitions. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $I = (x, k) \in \Sigma^* \times \mathbb{N}$, $k$ is called the *parameter*. A parameterized problem $L$ is

*fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$, a computable function $f$, and a constant $c$ such that given an instance $I = (x, k)$, $\mathcal{A}$ (called an FPT *algorithm*) correctly decides whether $I \in L$ in time bounded by $f(k) \cdot |I|^c$. For instance, the Vertex Cover problem parameterized by the size of the solution is FPT. A parameterized problem $L$ is in XP if there exists an algorithm $\mathcal{A}$ and two computable functions $f$ and $g$ such that given an instance $I = (x, k)$, $\mathcal{A}$ (called an XP *algorithm*) correctly decides whether $I \in L$ in time bounded by $f(k) \cdot |I|^{g(k)}$. For instance, the Clique problem parameterized by the size of the solution is in XP.

Within parameterized problems, the class W[1] may be seen as the parameterized equivalent to the class NP of classical decision problems. Without entering into details, a parameterized problem being W[1]-*hard* can be seen as a strong evidence that this problem is *not* FPT. The canonical example of W[1]-hard problem is Clique parameterized by the size of the solution.

If $\mathcal{B}$ is a collection of sets, for conciseness we use $\bigcup\mathcal{B}$ to denote the set $\bigcup_{A\in\mathcal{B}} A$. For a positive integer $k$, we denote by $[k]$ the set $\{1, \ldots, k\}$. For a sequence of sets $\mathcal{B} = (B_1, \ldots, B_k)$, a *transversal* of $\mathcal{B}$ is a set $\{b_1, \ldots, b_k\}$ such that $b_i \in B_i$ for all $i \in [k]$. Here we remark that the terms in $\mathcal{B}$ need not be distinct but the elements in a transversal $\{b_1, \ldots, b_k\}$ are distinct. For a set of indices $J$, we use $(B_j \mid j \in J)$ to denote the sequence of sets indexed by $J$ so we can use $\mathcal{B} = (B_j \mid j \in [k])$. A *subsequence* of $\mathcal{B}$ is a sequence $(B_j \mid j \in J)$ for $J \subseteq [k]$. A *partial transversal* of $\mathcal{B}$ is a transversal of some subsequence of $\mathcal{B}$. For convenience, we extend all notation regarding transversals and partial transversals to collections of sets.

If $P$ is a path in a digraph $G$, we denote by $\mathsf{s}^-(P)$ and $\mathsf{s}^+(P)$ the first and last vertices of $P$, respectively. Every vertex of $P$ other than $\mathsf{s}^-(P)$ and $\mathsf{s}^+(P)$ is an *internal* vertex. For $A, B \in V(G)$, we say that $P$ is an $A \to B$ *path* if $\mathsf{s}^-(P) \in A$ and $\mathsf{s}^+(P) \in B$. For $A, B \subseteq V(G)$ an $(A, B)$-*separator* is a set $X \subseteq V(G)$ such that there are no $A \to B$ paths in $G \setminus X$.

Let $\mathcal{P}$ be a collection of paths in $G$. We use $\mathsf{s}^-(\mathcal{P})$ to denote $\bigcup_{P\in\mathcal{P}} \mathsf{s}^-(P)$ and $\mathsf{s}^+(\mathcal{P})$ to denote $\bigcup_{P\in\mathcal{P}} \mathsf{s}^+(P)$. For conciseness, we say henceforth that the paths in $\mathcal{P}$ are *disjoint* if they are pairwise vertex-disjoint. For $A, B \in V(G)$, we say that $\mathcal{P}$ is a collection of $A \to B$ paths if $\mathsf{s}^-(\mathcal{P}) \subseteq A$ and $\mathsf{s}^+(\mathcal{P}) \subseteq B$. For the remaining of this article and unless stated otherwise, $n$ is used to denote the number of vertices of the input digraph of the problem under consideration.

▶ **Theorem 1** (Menger's Theorem [18]). *Let $G$ be a digraph and $A, B \subseteq V(D)$. The maximum size of a collection of disjoint $A \to B$ paths is equal to the minimum size of an $(A, B)$-separator. Furthermore, a maximum size collection of paths and a minimum size separator can be found in time $\mathcal{O}(n^2)$.*

A digraph $G$ is *strongly connected* if for every $u, v \in V(G)$ there are paths from $u$ to $v$ and from $v$ to $u$ in $G$. A *separator* of $G$ is a set $X \subseteq V(G)$ such that $G \setminus X$ has a single vertex or is *not* strongly connected. If $G$ has at least $k + 1$ vertices and $k$ is the minimum size of a separator of $G$, we say that $G$ is $k$-*strongly connected* (or $k$-*strong* for short). A *strongly connected component* (or *strong component* for short) of a digraph $G$ is a maximal induced subgraph of $G$ that is strongly connected.

The *directed tree-width* of digraphs was introduced by Johnson et al. [12] as a directed analogue of tree-width of undirected graphs. Informally, the directed tree-width $\mathsf{dtw}(G)$ of a digraph $G$ measures how close $G$ can be approximated by a DAG, and the formal definition immediately implies that $\mathsf{dtw}(G) = 0$ if and only if $G$ is an acyclic digraph (DAG). Directed tree-width and arboreal decompositions are not explicitly used in this article and thus we refer the reader to [12] for the formal definitions. Here it suffices to mention a few known results. In the same paper where they introduced directed tree-width, Johnson et al. [12] showed that $k$-DDP can be solved in XP time with parameters $k + \mathsf{dtw}(G)$.

▶ **Proposition 2** (Johnson et al. [12]). *The $k$-DDP problem is solvable in time $n^{\mathcal{O}(k+\mathsf{dtw}(G))}$.*

Notice that every instance of the $(k,c)$-DDP problem with $c \geq k$ is trivially solvable in polynomial time by simply checking for connectivity between each pair $(s_i, t_i)$. Thus we can always assume that $c < k$. It is easy to reduce the congested version to the disjoint version of DDP: it suffices to generate a new instance by making $c$ copies of each vertex $v$ of the input digraph, each of them with the same in- and out-neighborhood as $v$. A formal proof of this statement was given by Amiri et al. [1]. Hence it follows that $(k,c)$-DDP is also XP with parameters $k$ and $\mathsf{dtw}(G)$. A direct proof of this statement is also possible by applying the same framework used to prove Proposition 2, although such a proof is not given in [12]. A similar proof for a congested version of a DDP-like problem is given by Sau and Lopes in [16]. In any case, the following holds.

▶ **Proposition 3.** *The $(k,c)$-DDP problem is solvable in time $(c \cdot n)^{\mathcal{O}(c(k+\mathsf{dtw}(G)))}$.*

For both the $k$-DDP and $(k,c)$-DDP problems, (a small variation of) the result of Slivkins [19] implies that the XP time is unlikely to be improvable to FPT, even when restricted to DAGs (although the result in [19] concerns the *edge*-disjoint version of $k$-DDP, it easily implies W[1]-hardness of the disjoint version by noticing that the line digraph of a DAG is also a DAG).

As it is the case with tree-width, Johnson et al. [12] also introduced a dual notion for directed tree-width in the form of *havens*. However, although the duality in the undirected case is sharp, in the directed case it is only approximate: they showed that the directed tree-width of a digraph $G$ is within a constant factor (more precisely, a factor three) from the maximum order of a haven of $G$. Since havens and (strict) *brambles* are interchangeable in digraphs whilst paying only a constant factor for the transformation (see [6, Chapter 6] for example), we skip the definition of the former and focus only on the latter.

▶ **Definition 4** (Brambles in digraphs). *A* bramble $\mathcal{B} = \{B_1, \ldots, B_\ell\}$ *in a digraph $G$ is a collection of strong subgraphs of $G$ such that if $B, B' \in \mathcal{B}$ then $V(B) \cap V(B') \neq \emptyset$ or there are edges in $G$ from $V(B)$ to $V(B')$ and from $V(B')$ to $V(B)$. We say that the elements of $\mathcal{B}$ are the* bags *of $\mathcal{B}$. A* hitting set *of a bramble $\mathcal{B}$ is a set $C \subseteq V(G)$ such that $C \cap V(B) \neq \emptyset$ for all $B \in \mathcal{B}$. The* order *of a bramble $\mathcal{B}$, denoted by $\mathsf{ord}(\mathcal{B})$, is the minimum size of a hitting set of $\mathcal{B}$. A bramble $\mathcal{B}$ is said to be* strict *if for all pairs $B, B' \in \mathcal{B}$ it holds that $V(B) \cap V(B') \neq \emptyset$. For an integer $c \geq 1$ we say that $\mathcal{B}$ has* congestion *$c$ if every vertex of $G$ appears in at most $c$ bags of $\mathcal{B}$.*

See Figure 1 for an example of a bramble. Notice that if $\mathcal{B}$ is a bramble of congestion $c$ for some constant $c$, its *order* increases together with its *size*; i.e., $|\mathcal{B}|$. More precisely, since every vertex of the host digraph can hit at most $c$ elements of $\mathcal{B}$ it holds that $\mathsf{ord}(\mathcal{B}) \geq \lceil |\mathcal{B}|/c \rceil$. If $\mathcal{B}' \subseteq \mathcal{B}$ then we may say that $\mathcal{B}'$ is a *subbramble* of $\mathcal{B}$.

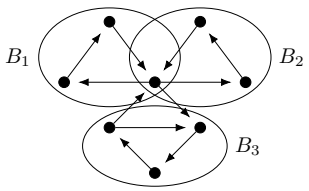



**Figure 1** Example of a bramble $\{B_1, B_2, B_3\}$ of order two.

Johnson et al. [12] gave an algorithm that, given a digraph $G$, either correctly decides that $\mathsf{dtw}(G) \leq 3k-2$ (also yielding an arboreal decomposition of $G$) or produces a bramble of order $\lfloor k/2 \rfloor$. This was later improved to an FPT algorithm by Campos et al. [4]. Although the authors do not explicitly say that the produced bramble is strict, it is easy to verify that this is the case in their proof, and the same holds for the proof of [12].

▶ **Proposition 5** (Campos et al. [4]). *Let $G$ be a digraph and $t$ be a non-negative integer. There is an algorithm running in time $2^{\mathcal{O}(t \log t)} \cdot n^{\mathcal{O}(1)}$ that either produces an arboreal decomposition of $G$ of width at most $3t-2$ or finds a strict bramble of order $t$ in $G$.*

Brambles of constant congestion are a key structure used to solve instances of $(k,c)$-DDP in $f(k)$-strong digraphs in the approach by Edwards et al. [8], as discussed in Section 4. In particular, they use the following result, originally proved by Kawarabayashi and Kreutzer [14] where an XP algorithm is given, and then improved by Campos et al. [4] with an FPT algorithm and a better dependency on $k$. We refer the reader to [4, 14] for the definition of well-linked sets, and we remark that, for convenience, we present the statement of the following result in a slightly different way than in the original article.

▶ **Proposition 6** (Campos et al. [4]). *Let $g(k) = (t+1)(\lfloor t/2 \rfloor + 1) - 1$ and $G$ be a digraph with directed tree-width at least $3g(t)-1$. There is an algorithm running in time $2^{\mathcal{O}(t^2 \log t)} \cdot n^{\mathcal{O}(1)}$ that finds in $G$ a bramble $\mathcal{B}$ of order $g(t)$, a path $P$ that intersects every bag of $\mathcal{B}$, and a well-linked set $X$ of size $t$ such that $X \subseteq V(P)$.*

▶ **Proposition 7** (Edwards et al. [8]). *There exists a function $f : \mathbb{N} \to \mathbb{N}$ satisfying the following. Let $G$ be a digraph and $t \geq 1$ be an integer. Let $P$ be a path in $G$ and $X \subseteq V(P)$ be a well-linked set with $|X| \geq f(t)$. Then $G$ contains a bramble $\mathcal{B}$ of congestion two and size $t$ and, given $G, P$, and $X$, we can find $\mathcal{B}$ in polynomial time.*

Pipelining Propositions 5–7 we obtain the following.

▶ **Corollary 8.** *There is a function $f : \mathbb{N} \to \mathbb{N}$ and an FPT algorithm with parameter $t$ that, given a digraph $G$ and an integer $t \geq 1$, either correctly decides that the directed tree-width of $G$ is at most $f(t)$ or finds a bramble $\mathcal{B}$ of congestion two and size $t$ in $G$.*

## 3 New Menger-like statements for paths and cuts in digraphs

In this section we present the definition and the min-max formulas associated with each pair D-paths/D-cuts, T-paths/T-cuts, and R-paths/R-cuts. Since all three types of paths share some properties (in fact, the major distinction between them is in how they reach their destinations), it is convenient to adopt the following notations.

▶ **Definition 9** (Digraph-source sequences and respecting paths). *For an integer $\ell \geq 1$, a* digraph-source sequence *of* size $\ell$ *is a pair $(\mathcal{F}, \mathcal{S})$ such that $\mathcal{F} = (G_1, \ldots, G_\ell)$ is a sequence of digraphs and $\mathcal{S} = (S_1, \ldots, S_\ell)$ is a sequence of subsets of vertices with $S_i \subseteq V(G_i)$ for $i \in [\ell]$. We say that a set of paths $\mathscr{P}$* respects *$(\mathcal{F}, \mathcal{S})$ or, alternatively, is $(\mathcal{F}, \mathcal{S})$*-respecting *if there is a partition $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ of $\mathscr{P}$ such that*
**(a)** *for $i \in [\ell]$, $\mathcal{P}_i$ is a set of disjoints paths in $G_i$, and*
**(b)** *for $i \in [\ell]$, $\mathsf{s}^-(\mathcal{P}_i) \in S_i$.*
*In this case, we say that $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ is a* defining partition *of $\mathscr{P}$.*

Thus in any set of $(\mathcal{F}, \mathcal{S})$-respecting paths, any two paths can intersect only if they belong to distinct parts of the defining partition.

To provide some intuition within the context of the $(k, c)$-DDP problem, in the next three definitions one can think of the sequence $(S_1, \ldots, S_\ell)$ as being formed by many copies of $S$ followed by many copies of $T$. Notice that any set of $(\mathcal{F}, \mathcal{S})$-respecting paths includes paths *leaving* $T$, which seems counter-intuitive when considering the goal of solving instances of $(k, c)$-DDP. We remark this can be easily addressed by associating, with each copy of $T$ in the sequence $\mathcal{S}$, the digraph $G^{\mathsf{rev}}$ obtained by reversing the orientation of every edge of $G$.

**D-paths and D-cuts.** Within the context of this paper, D-paths and D-cuts are used as a tool to prove our results regarding T-paths and T-cuts. In this scenario, one should think of the set $B$ in the following definition as the set of vertices of a highly connected structure that is intended to be used to appropriately connect the last vertices of paths from $S$ to the first vertices of paths from $T$.

▶ **Definition 10** (D-paths and D-cuts). *For an integer $\ell \geq 1$, let $(\mathcal{F}, \mathcal{S})$ be a digraph-source sequence with $\mathcal{F} = (G_1, \ldots, G_\ell)$, $\mathcal{S} = (S_1, \ldots, S_\ell)$, and $B \subseteq \bigcup_{i=1}^{\ell} V(G_i)$. With respect to $B$, we say that a set of $(\mathcal{F}, \mathcal{S})$-respecting paths $\mathscr{P}$ with defining partition $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ is a set of* D-paths *if*

**(1)** *for all distinct $P, P' \in \mathscr{P}$ it holds that $\mathsf{s}^+(P) \neq \mathsf{s}^+(P')$, and*

**(2a)** $\mathsf{s}^+(\mathscr{P}) \subseteq B$.

*A* D-cut *is a sequence $\mathcal{X} = (X_0, \ldots, X_\ell)$ with $X_0 \subseteq B$ such that, for $i \in [\ell]$, the set $X_i \subseteq V(G_i)$ is an $(S_i, B \setminus X_0)$-separator in $G_i$. The* order *of a D-cut $X$ is $\mathsf{ord}(\mathcal{X}) = |X_0| + \sum_{i=1}^{\ell} |X_i|$.*

Thus, in the definition of D-paths we ask each collection of paths associated with each $\mathcal{P}_i$ to be pairwise disjoint in $G_i$, and the paths from distinct parts $\mathcal{P}_i, \mathcal{P}_j$ may share vertices in $\bigcup_{i=1}^{\ell} V(G_i)$ other than the last vertices of the paths. The "D" in the name stands for "disjoint". For the min-max formula, we prove the following.

▶ **Theorem 11** ($\star$). *Let $(\mathcal{F}, \mathcal{S})$ be digraph-source sequence of size $\ell$ with $\mathcal{F} = (G_1, \ldots, G_\ell)$, and let $B \subseteq \bigcup_{i=1}^{\ell} V(G_i)$. With respect to $\mathcal{F}, \mathcal{S}$, and $B$, the maximum number of D-paths is equal to the minimum order of a D-cut. Additionally, a D-cut of minimum order and a maximum collection of D-paths can be found in time $\mathcal{O}((\ell \cdot n^* + |\mathcal{B}|)^2)$ where $n^* = \max_{i \in [\ell]}(|V(G_i)|)$.*

**T-paths and T-cuts.** For the sake of intuition, in the next definition one should think of $\mathcal{B}$ as a bramble. Informally, and given a digraph $G$, we use T-paths and T-cuts to find a large collection of paths from a given ordered $S \subseteq V(G)$ to the bags of a subbramble $\mathcal{B}^S \subseteq \mathcal{B}$, and from the bags of another subbramble $\mathcal{B}^T \subseteq \mathcal{B}$ to $T$ (we can achieve this orientation for these paths by reversing the orientation of the edges of $G$), while ensuring that every vertex outside of $\mathcal{B}$ appears in at most two of those paths, and that all elements of $\mathcal{B}^S \cup \mathcal{B}^T$ are pairwise distinct. The first property can be achieved by simply applying Menger's Theorem (cf. Theorem 1) twice. However, by doing this, we can end with a set of paths all ending on the same bag of $\mathcal{B}$. This scenario is far from ideal, since at some point the goal is to use the strong connectivity of $G[B \cup B']$ for every $B, B' \in \mathcal{B}$ to appropriately connect the ending vertices of the paths from $S$ to the starting vertices of the paths to $T$, while maintaining the property that every vertex appears in at most two (or $c$, in the general case) of those paths. If a unique bag $B$ contains all starting and ending vertices of the paths, then connecting those vertices while maintaining such properties may be as hard as finding a solution to an instance of $(k, c)$-DDP in the strong digraph $G[B]$, or downright impossible to do.

Therefore, in the proofs applying similar techniques, as seen in the works by Edwards et al. [8] and by Giannopoulou et al. [10], there is considerable effort into finding paths with "good properties" that can be used to connected the paths inside of $\mathcal{B}$ (or inside of a

cylindrical grid in the case of [10]), and these properties always include, as far as we know, that the paths end or start in distinct bags of $\mathcal{B}$ (or distinct sections of the cylindrical grid). In particular, [8, Lemma 16] includes a set of seven properties over a set of paths that we can substitute by T-paths to achieve better results in a simpler manner.

We can prove the same results using R-paths and R-cuts, which are simpler than T-paths and T-cuts. We include the proofs for the two latter objects for potential applications in which the extra properties of T-paths and T-cuts may become handy.

▶ **Definition 12** (T-paths and T-cuts). *For an integer $\ell \geq 1$, let $(\mathcal{F}, \mathcal{S})$ be a digraph-source sequence with $\mathcal{F} = (G_1, \ldots, G_\ell)$ and $\mathcal{S} = (S_1, \ldots, S_\ell)$, and $\mathcal{B}$ be a family of subsets of $\bigcup_{i=1}^{\ell} V(G_i)$. With respect to $\mathcal{B}$, we say that a set of $(\mathcal{F}, \mathcal{S})$-respecting paths $\mathscr{P}$ with defining partition $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ is a set of* T-paths *if*
**(1)** *for all distinct $P, P' \in \mathscr{P}$ it holds that $\mathsf{s}^+(P) \neq \mathsf{s}^+(P')$, and*
**(2b)** *the set $\mathsf{s}^+(\mathscr{P})$ is a partial transversal of $\mathcal{B}$.*
*A* T-cut *is a pair $(\mathcal{B}', \mathcal{X})$ with $\mathcal{B}' \subseteq \mathcal{B}$ and such that $\mathcal{X}$ is a D-cut with respect to $\mathcal{F}, \mathcal{S}$, and $\bigcup\mathcal{B}'$. The* order *of a T-cut $(\mathcal{B}', X)$ is $\mathsf{ord}(\mathcal{B}', X) = |\mathcal{B} \setminus \mathcal{B}'| + \mathsf{ord}(\mathcal{X})$.*

For convenience, we keep only one set of parenthesis, writing $\mathsf{ord}(\mathcal{B}', \mathcal{X})$ instead of $\mathsf{ord}((\mathcal{B}', \mathcal{X}))$.

Notice that conditions **(1)** in the definition of T-paths is the same as in the definition of D-paths. Thus the difference between D-paths and T-paths is that in the former we ask the paths to end in distinct vertices of $B$, while in the latter we ask the endpoints of the paths to form a partial transversal of $\mathcal{B}$. This implies that those endpoints are distinct, and that each of them is associated with a unique element of $\mathcal{B}$. The "T" in the name stands for "transversal". See Figure 2 for an example of a transversal of a collection of sets.

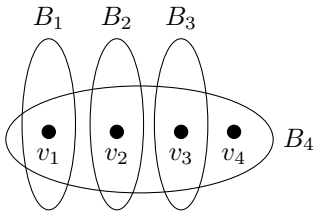



**Figure 2** Example of a transversal of the collection $\{B_1, B_2, B_3, B_4\}$. For $i \in [4]$ the vertex $v_i$ is associated with the set $B_i$.

For the T-paths/T-cuts duality, we prove the following.

▶ **Theorem 13** (⋆). *Let $(\mathcal{F}, \mathcal{S})$ be a digraph-source sequence of size $\ell$ with $\mathcal{F} = (G_1, \ldots, G_\ell)$, and let $\mathcal{B}$ be a collection of subsets of $\bigcup_{i=1}^{\ell} V(G_i)$. With respect to $\mathcal{F}, \mathcal{S}$, and $\mathcal{B}$, the maximum number of T-paths is equal to the minimum order of a T-cut. Additionally, a T-cut of minimum order and a maximum collection of T-paths can be found in time $\mathcal{O}((\ell \cdot n^* + |\mathcal{B}|)^2)$, where $n^* = \max_{i \in [\ell]}(|V(G_i)|)$.*

**R-paths and R-cuts.** The intuition for R-paths is similar to the one for T-paths, as is the motivation to use these objects in the context of $(k, c)$-DDP and similar problems. The difference between them is that, if $\mathscr{P}$ is a set of T-paths, then all vertices of the form $\mathsf{s}^+(P)$, where $P \in \mathscr{P}$, are distinct. In R-paths this only holds when considering paths inside of the same part of its defining partition. More precisely, given a partition $\mathcal{P}$ of a set of R-paths as defined below, $\mathsf{s}^+(P)$ and $\mathsf{s}^+(P')$ are guaranteed to be disjoint only when $P$ and $P'$ are in distinct parts of $\mathcal{P}$. In Section 4 we show that this relaxation poses no problem for the application of R-paths/R-cuts we show in this article.

▶ **Definition 14** (R-paths and R-cuts). *For $\ell \geq 1$, let $(\mathcal{F}, \mathcal{S})$ be a digraph-source sequence with $\mathcal{F} = (G_1, \ldots, G_\ell)$ and $\mathcal{S} = (S_1, \ldots, S_\ell)$, and $\mathcal{B}$ be a family of subsets of $\bigcup_{i=1}^{\ell} V(G_i)$. With respect to $\mathcal{B}$, we say that a set of $(\mathcal{F}, \mathcal{S})$-respecting paths $\mathscr{P}$ with defining partition $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ is a set of* R-paths *if*

**(1c)** *for some family $\mathcal{B}^* \subseteq \mathcal{B}$ there is a bijective mapping $h : \mathscr{P} \to \mathcal{B}^*$ such that $h(P) = B$ implies $\mathsf{s}^+(P) \in B$.*

*An* R-cut *is a pair $(\mathcal{B}', \mathcal{X})$ where $\mathcal{B}' \subseteq \mathcal{B}$ and $\mathcal{X}$ is a sequence $(X_1, \ldots, X_\ell)$ such that each $X_i \in \mathcal{X}$ is an $(S_i, \bigcup \mathcal{B}')$-separator in $G_i$. The* order *of an R-cut $(\mathcal{B}', \mathcal{X})$ is $\mathsf{ord}(\mathcal{B}', \mathcal{X}) = |\mathcal{B} \setminus \mathcal{B}'| + \sum_{i=1}^{\ell} |X_i|$.*

We remark that the only difference between R-paths and T-paths is that, in the latter, condition **(1)** ensures that all vertices forming the partial transversal of $\mathcal{B}$ are distinct. The "R" in the name stands for "representatives". For the duality, we prove the following.

▶ **Theorem 15** (⋆). *Given a digraph-source sequence $(\mathcal{F}, \mathcal{S})$ of size $\ell$ and a set $B \subseteq \bigcup_{i=1}^{\ell} V(G_i)$ then, with respect to $\mathcal{F}, \mathcal{S}$, and $B$, the maximum number of R-paths is equal to the minimum order of an R-cut. Additionally, an R-cut of minimum order and a maximum collection of R-paths can be found in $\mathcal{O}((k \cdot n^* + |\mathcal{B}|)^2)$ where $n^* = \max_{i \in [\ell]}(|V(G_i)|)$.*

Observe that the right side of the pair forming an R-cut cannot be simply a set of vertices $X$ because, for example, a vertex $v \in X$ can be in two distinct $G_i$ and $G_j$ and be part of the separator in $G_i$ but *not* part of the separator in $G_j$. In this case $v$ would be counted twice in the order of the R-cut, but it is only used in one separator. Also notice that when $|\mathcal{B}|$ is larger than the allowed budget to construct an R-cut, every R-cut of appropriate order must identify some separator in some $G_i$, i.e., $\mathcal{X} \neq \emptyset$. In fact, there are only two options for the size of $\mathcal{X}$: either $\mathcal{B}' = \emptyset$ and hence $\mathcal{X} = \emptyset$, or $\mathcal{B}' \neq \emptyset$ and $|\mathcal{X}| = \ell$. In the latter case, it is possible that some $X_i \in \mathcal{X}$ are empty. In Lemma 19 we exploit this fact to show how to either find in a digraph $G$ a large collection of R-paths from a set $S$ to the vertices appearing in the elements of some sufficiently large collection $\mathcal{B}$ (corresponding to a bramble), or a small separator intersecting every path from $S$ to all such vertices.

## 4 Applications

In this section we show how to exploit the duality between R-paths and R-cuts to improve on results by Edwards et al. [8] and Giannopoulou et al. [11]. The following is the main result that we prove, and then we use it to improve on results by [8, 11].

▶ **Theorem 16.** *Let $k, c$ be integers with $k, c \geq 2$ and $g(k, c) = 2k(c \cdot k - c + 2) + c(k - 1)$. Let $G$ be a digraph, assume that we are given the bags of a bramble $\mathcal{B}$ of congestion $c$ and size at least $g(k, c)$, and $S, T \subseteq V(G)$ with $S = \{s_1, \ldots, s_k\}$ and $T = \{t_1, \ldots, t_k\}$. Then in time $\mathcal{O}(k^4 \cdot n^2)$ one can either*

1. *find a $\mathcal{B}^* \subseteq \mathcal{B}$ with $|\mathcal{B}^*| \geq g(k, c) - c(k-1)$ and an $(S, \bigcup \mathcal{B}^*)$-separator $X_S$ with $|X_S| \leq k-1$ that is disjoint from all bags of $\mathcal{B}^*$, or*
2. *find a $\mathcal{B}^* \subseteq \mathcal{B}$ with $|\mathcal{B}^*| \geq g(k, c) - c(k-1)$ and an $(\bigcup \mathcal{B}^*, T)$-separator $X_T$ with $|X_T| \leq k-1$ that is disjoint from all bags of $\mathcal{B}^*$, or*
3. *find a set of paths $\{P_1, \ldots, P_k\}$ in $G$ such that each $P_i$ with $i \in [k]$ is a path from $s_i$ to $t_i$ and each vertex of $G$ appears in at most $c$ of these paths.*

Theorem 16 yields an XP algorithm with parameter $k$ for the $(k, c)$-DDP problem in $k$-strong digraphs, as we proceed to discuss. First, we remark that the XP time is only required when a large bramble of congestion at most $c$ is not provided. If this is the case, we

first look at the directed tree-width of $G$, which can be approximated in FPT time applying Proposition 5. If $\mathsf{dtw}(G) \leq f(k)$ for some computable function $f$, then we solve the problem applying Proposition 3. Otherwise, we apply the machinery by Edwards et al. [8] pipelining Propositions 6 and 7 to obtain a large bramble of congestion two in digraphs of large directed tree-width, and use Theorem 16 to find a solution in polynomial time, which we show to always be possible. When assuming that the input digraph is $k$-strong, only the third output of Theorem 16 is possible, and therefore as a direct consequence of Theorem 16 we obtain the following.

▶ **Theorem 17.** *Let $G$ be a $k$-strong digraph and $\mathcal{B}$ be a bramble of congestion $c \geq 2$ with $|\mathcal{B}| \geq 2k(c \cdot k - c + 2) + c(k-1)$. Then for any ordered sets $S, T \subseteq V(G)$ both of size $k$, the instance $(G, S, T)$ of $(k,c)$-DDP with $c \geq 2$ is positive and a solution can be found in time $\mathcal{O}(k^4 \cdot n^2)$.*

As mentioned in the introduction, our result improves over the result of Edwards et al. [8] by relaxing the strong connectivity of the input digraph from $36k^3 + 2k$ to $k$ (and this bound is close to the best possible unless P = NP), by needing a smaller bramble (from size $188k^3$ to $4k^2 + 2k - 2$ when $c = 2$), and because the proof is simpler and shorter. Finally, applying Corollary 8, Proposition 3, and Theorem 17 (thus again using Proposition 7 by [8]) we immediately obtain the following.

▶ **Corollary 18.** *For every integer $c \geq 2$, the $(k,c)$-DDP problem is solvable in XP time with parameter $k$ in $k$-strong digraphs.*

As a tool to prove Theorem 16, we first show how we can take many copies of digraphs $G$ and $G'$, say $\ell$ copies of each, to either find $2\ell \cdot k$ R-paths to a given collection $\mathcal{B}$ of sufficiently large size, or find an appropriate separator of size at most $k-1$ in $G$ or in $G'$.

▶ **Lemma 19.** *Let $(\mathcal{F}, \mathcal{S})$ be a digraph-source sequence where $\mathcal{F}$ contains $\ell$ copies of a digraph $G_S$ and $\ell$ copies of a digraph $G_T$, in this order, and $\mathcal{S}$ contains $\ell$ copies of a set $S \subseteq V(G_S)$ and $\ell$ copies of a set $T \subseteq V(G_T)$, in this order, where $|S| = |T| = k$. Finally, let $\mathcal{B}$ be a collection of subsets of $V(G)$ with $|\mathcal{B}| \geq 2\ell \cdot k$. Then either there is a set of R-paths with respect to $\mathcal{F}, \mathcal{S}$, and $\mathcal{B}$ of size at least $2\ell \cdot k$, or for some non-empty $\mathcal{B}' \subsetneq \mathcal{B}$ there is an R-cut $(\mathcal{B}', \mathcal{X})$ of order at most $2\ell \cdot k - 1$ such that $\mathcal{X}$ contains $\ell$ copies of an $(S, \bigcup \mathcal{B}')$-separator $X_S$ followed by $\ell$ copies of an an $(T, \bigcup \mathcal{B})$-separator $X_T$ with $|X_S| + |X_T| \leq 2k - 1$.*

**Proof.** Assume that the maximum size of a set of R-paths with respect to $\mathcal{F}, \mathcal{S}$, and $\mathcal{B}$ is at most $2\ell \cdot k - 1$. Then by Theorem 15 there is a minimum R-cut $Y = (\mathcal{B}', \mathcal{X}')$ of order at most $2\ell \cdot k - 1$.

Since $|\mathcal{B}| \geq 2\ell \cdot k$ we conclude that $\mathcal{B}' \neq \emptyset$ and thus $|\mathcal{X}'| = 2\ell$. (we refer the reader to the discussion in the end of the first part of Section 3). Hence by the choice of $\mathcal{F}$ and $\mathcal{S}$ and the definition of R-cuts, we can construct an R-cut $(\mathcal{B}', \mathcal{X})$ with the same order as $Y$ by simply including in $\mathcal{X}$ exactly $\ell$ copies of the $(S, \bigcup \mathcal{B}')$-separator $X_S$ contained in $\mathcal{X}'$, and $\ell$ copies of the $(T, \bigcup \mathcal{B}')$-separator $X_T$ contained in $\mathcal{X}'$. Thus the newly generated R-cut satisfies $|\mathcal{B} \setminus \mathcal{B}'| + \ell(|X_S| + |X_T|) = \mathsf{ord}(\mathcal{B}', \mathcal{X}) = \mathsf{ord}(\mathcal{B}', \mathcal{X}') \leq 2\ell k - 1$. This immediately implies that $|X_S| + |X_T| \leq 2k - 1$ and the result follows. ◀

Now the plan is to apply twice the duality between R-paths and R-cuts. In the first application, we consider the digraph-source sequence formed by $2k(c \cdot k - c + 1)$ copies of $G_S = G$ and then exactly as many copies of $G_T = G^{\mathsf{rev}}$, and the same number of copies of $S$ and $T$, also in this order. If we do not find a set of $2k(c \cdot k - c + 1)$ R-paths to the bags of the

bramble, then we apply Lemma 19 and find an R-cut $(\mathcal{B}', \mathcal{X})$ of small order and a separator of size at most $k-1$ in $G_S$ or $G_T$. Since the given bramble has congestion $c$, we show that we can stop with output **1** or **2** of Theorem 16 by taking the bramble $\mathcal{B}^*$ containing all bags of $\mathcal{B}$ that are disjoint from the small separator. This holds because no bag of $\mathcal{B}^*$ can be in the "wrong side" of the separator. That is, if for instance there is an $S \to A$ path $P$ avoiding the separator $X_S$ of size at most $k-1$ in $G_S$, then every bag of $\mathcal{B}^*$ must be in $\mathcal{B} \setminus \mathcal{B}'$ since otherwise one can construct a path in $G_S \setminus X_S$ from $S$ to a bag of $\mathcal{B}'$ by using $P$ and the connectivity properties of the bramble. In this case, we also arrive at a contradiction since the size of $\mathcal{B}^*$ is too large to be contained in the R-cut.

If the R-paths are found, then we refine the digraph by carefully choosing edges to delete from $G$ in such a way that, from a second application of the duality between R-paths and R-cuts, we are guaranteed to find $2k$ R-paths that can be used to construct the $\{s_i\} \to \{t_i\}$ paths while maintaining the congestion under control. In the refined digraph, we keep the R-paths found in the first iteration and delete edges leaving vertices of the bramble appearing in bags that were not used as destinations for the R-paths. We apply Lemma 19 in this digraph and show that the only possible outcome is that the R-paths are found. Otherwise, there is an R-cut of order at most $2k-1$ and hence there is a $\mathcal{B}'' \subseteq \mathcal{B}$ that contains at least one bag $A$ that is not in the R-cut and is disjoint from the separator, say $X'_S$, of size at most $k-1$ that is part of the R-cut. Now the size of $X'_S$ implies that there is a path from $S$ to a bag disjoint from $X'_S$ avoiding the separator, and thus we can reach $A$ from $S$ avoiding $X'_S$. Finally, the refined digraph allows us to associate each vertex of the bramble used by a path in $\{P_1, \ldots, P_k\}$ with a bag of the bramble, depending on where the vertex appears in the path, in such a way that no vertex is associated twice with the same bag by two distinct paths. Together with the bound on the congestion of the bramble, this immediately implies that every vertex of the digraph appears in at most $c$ paths of the set $\{P_1, \ldots, P_k\}$.

**Proof of Theorem 16.** Let $G$, $S$, $T$, and $\mathcal{B}$ be as in the statement of the theorem. Define $G_S = G$ and $G_T = G^{\mathsf{rev}}$ and let $(\mathcal{F}, \mathcal{S})$ be a digraph-source sequence with $\mathcal{F}$ containing, in order, $c \cdot k - c + 1$ copies of $G_S$ followed by $c \cdot k - c + 1$ copies of $G_T$, and $\mathcal{S}$ containing, in order, the same number of copies of $S$ followed by exactly as many copies of $T$. Now, applying Lemma 19 with respect to $\mathcal{F}, \mathcal{S}, \mathcal{B}$, and $\ell = c \cdot k - c + 1$, we conclude that either there are $2k(c \cdot k - c + 1)$ R-paths or, for some $\mathcal{B}' \subseteq \mathcal{B}$ there is an R-cut $(\mathcal{B}', \mathcal{X})$ where $\mathcal{X}$ contains $\ell$ copies of an $(S, \bigcup \mathcal{B}')$-separator $X_S$ and $\ell$ copies of an an $(S, \bigcup \mathcal{B}')$-separator $X_T$ with $|X_S| + |X_T| \leq 2k - 1$. We first consider the case where the separators are obtained. Thus $|X_S| \leq k-1$ or $|X_T| \leq k-1$. Since both cases are symmetric (the $T \to X_T$ paths become $X_T \to T$ paths when we restore the orientation of the edges of $G_T$), we suppose without loss of generality that $|X_S| \leq k-1$.

Let $\mathcal{B}^*$ contain all bags of $\mathcal{B}$ that are disjoint from $X_S$. Since $\mathcal{B}$ has congestion $c$ we conclude that $|\mathcal{B}^*| \geq g(k,c) - c(k-1)$. We show that no vertex appearing in a bag of $\mathcal{B}^*$ is reachable from $S$ in $G_S \setminus X_S$. By contradiction, assume that there is an $S \to A$ path $P$ in $G_S \setminus X_S$ for some $A \in \mathcal{B}^*$. If there is $A' \in \mathcal{B}^* \cap \mathcal{B}'$ then we can use the strong connectivity of $G_S[A \cup A']$ and the path $P$ to construct an $S \to A'$ path in $G_S \setminus X_S$, contradicting the choice of $X_S$. Thus in this case $\mathcal{B}^*$ must be entirely contained in $\mathcal{B} \setminus \mathcal{B}'$. Again we obtain a contradiction since $2k(c \cdot k - c + 2) \leq |\mathcal{B}^*| \leq |\mathcal{B} \setminus \mathcal{B}'| \leq \mathsf{ord}(\mathcal{B}', \mathcal{X}) < 2k(c \cdot k - c + 1)$. In other words, the existence of path from $S$ to a vertex in a bag of $\mathcal{B}^*$ avoiding $X_S$ implies that every bag of $\mathcal{B}^*$ is reachable from $S$ in $G_S \setminus X_S$ and thus such path cannot exist since $\mathcal{B}^*$ is too large to be contained in any R-cut of order less than $2k(c \cdot k - c + 1)$. We conclude that no bag $A \in \mathcal{B}^*$ is reachable from $S$ in $G_S \setminus X_S$, and and output **1** of the theorem follows. Symmetrically, output **2** of the theorem follows if $|X_T| \leq k-1$.

Assume now that a set of R-paths $\mathscr{P}$ of size at least $2k(c \cdot k - c + 1)$ is found. Let $\mathcal{B}^1 \subseteq \mathcal{B}$ and let $h : \mathscr{P} \to \mathcal{B}^1$ be the bijective mapping as in **(1c)** of Definition 14. Thus $h(p) = B$ implies that $\mathsf{s}^+(P) \in B$. Since $|S| = |T| = k$, we can split $\mathscr{P}$ into two sets of equal size $\mathscr{P}^S$ and $\mathscr{P}^T$, where every path in $\mathscr{P}^S$ is a path leaving $S$ in $G_S$ and every path in $\mathscr{P}^T$ is a path leaving $T$ in $G_T$. We define $\mathcal{B}_S = \{h(P) \in \mathcal{B}^1 \mid P \in \mathscr{P}^S\}$ and $\mathcal{B}_T = \{h(P) \in \mathcal{B}^1 \mid P \in \mathscr{P}^T\}$. The next step is to refine $\mathscr{P}$ to make it *minimal with respect to* $\mathcal{B}$. That is, we say that $\mathscr{P}$ is $\mathcal{B}$-*minimal* if no path of $\mathscr{P}$ contains an internal vertex that is in a bag $\mathcal{B} \setminus (\mathcal{B}_S \cup \mathcal{B}_T)$. In other words, when following a path $P \in \mathscr{P}$ from the first to the last vertex, if we find an internal vertex $v$ in a bag $B$ that is not in $\mathcal{B}_S$ nor in $\mathcal{B}_T$, we swap $P$ in $\mathscr{P}$ by its subpath ending in $v$ and update $\mathcal{B}_S$ or $\mathcal{B}_T$ accordingly. Clearly, condition **(1c)** of Definition 14 still hold with respect to the new choice of $\mathscr{P}$, and therefore from now on we assume that $\mathscr{P}$ is $\mathcal{B}$-minimal. This property is important later to bound the maximum number of times that a vertex can appear in the $S \to T$ paths we construct.

Next, we reduce $G_S$ and $G_T$ to digraphs that are still well connected to $\mathcal{B}$, thus ensuring that we can find a large set of R-paths in the new digraphs, in which we can maintain control over how many times a vertex can appear in the $S \to T$ paths we construct from these new R-paths. To this end, define

$$V_S = \bigcup_{P \in \mathscr{P}^S} (V(P) \setminus \{\mathsf{s}^+(P)\}) \cup \bigcup_{A \in \mathcal{B}_S} A \quad \text{and} \quad V_T = \bigcup_{P \in \mathscr{P}^T} (V(P) \setminus \{\mathsf{s}^+(P)\}) \cup \bigcup_{A \in \mathcal{B}_T} A. \tag{1}$$

Finally, we construct the digraphs $G'_S$ and $G'_T$ starting from $G_S$ and $G_T$, respectively, applying the following two rules:

- For every $v \in V(G_S)$, if $v \in (\bigcup \mathcal{B}) \setminus V_S$ then we delete from $G'_S$ every edge leaving $v$.
- For every $v \in V(G_T)$, if $v \in (\bigcup \mathcal{B}) \setminus V_T$ then we delete from $G'_T$ every edge leaving $v$.

Consider the digraph-source sequence $(\{G'_S, G'_T\}, \{S, T\})$ and let $\mathcal{B}' = \mathcal{B} \setminus (\mathcal{B}_S \cup \mathcal{B}_T)$, and notice that $\mathcal{B}'$ may not be a bramble in $G'_S$ nor in $G'_T$. Clearly $|\mathcal{B}'| \geq g(k,c) - 2k(c \cdot k - c + 1) = c(k-1) + 2k > 2k$. We apply Lemma 19 with respect to $\{G'_S, G'_T\}, \{S, T\}$ (and thus $\ell = 1$), and $\mathcal{B}'$, to either obtain a set of R-paths $\mathscr{P}'$ of size at least $2k$ or an R-cut $(\mathcal{B}'', \{X'_S, X'_T\})$ with order at most $2k-1$ where $|X'_S| + |X'_T| \leq 2k-1$ and $\mathcal{B}'' \neq \emptyset$. We claim that only the first output is possible. By contradiction, assume that the R-cut and the separators were obtained and, without loss of generality, that $|X_S| \leq k-1$. First notice that the upper bound on the order of the R-cut implies that $|\mathcal{B}'' \setminus \mathcal{B}'| \geq c(k-1) + 1$. Since $|X_S| \leq k-1$ this implies that there is at least one bag $A' \in \mathcal{B}''$ that is disjoint from $X_S$ and not included in R-cut.

Now, set $q = 2(c \cdot k - c + 1)$ and let $\mathcal{P}_1, \ldots, \mathcal{P}_q$ be the defining partition of $\mathscr{P}$. That is, for every $i \in [q]$, the part $\mathcal{P}_i$ is a set of $k$ disjoint paths in the $i$-th digraph of $\mathcal{F}$ (this is possible since $|S| = |T| = k$ and hence $\mathscr{P}$ cannot contain more than $k$ disjoint paths in any digraph in $\mathcal{F}$). Thus exactly $q/2$ parts $\mathcal{P}_i$ contain only paths starting in $S$. Now the size of $X_S$ allows it to intersect at most $c(k-1)$ bags of $\mathcal{B}$, and thus for some $\mathcal{P}_i$ no bag in $\mathcal{B}_i = \{A \in \mathcal{B}_S \mid P \in \mathcal{P}_i \text{ and } \mathsf{s}^+(P) \in A\}$ is intersected by $X'_S$. Since $|X'_S| \leq k-1$, there is a $P \in \mathcal{P}_i$ from $S$ to a bag $A \in \mathcal{B}_i$ that is not intersected by $X'_S$. By the choice of $G'_S$ this path also exists in this digraph and, since $\mathsf{s}^+(P) \in A$ and $A \in \mathcal{B}_S$, every edge of $G_S$ leaving every vertex in $A$ is kept in $G'_S$ and thus $G'_S[A]$ is strong. Now, as $\mathcal{B}$ is a bramble, we can construct a path from $S$ to $A'$ (remember that $A' \in \mathcal{B}''$ and $A' \cap X_S = \emptyset$) by following the path $P$ and then taking a path from $\mathsf{s}^+(P)$ to $A'$ in $G'_S[A \cup A']$, which in turn is guaranteed to exist since either $A \cap A' \neq \emptyset$ or there is an edge from $A$ to $A'$ in $G'_S$. This contradicts our assumption that $(\mathcal{B}'', \{X'_S, X'_T\})$ is an R-cut and the claim follows.

Assume now that a set $\mathscr{P}'$ of $2k$ R-paths is obtained. Therefore, there are disjoint paths $\{Q_1^S, \ldots, Q_k^S\}$ leaving $S$ and disjoint paths $\{Q'_1, \ldots, Q'_k\}$ leaving $T$ in $\mathscr{P}$. By recovering the orientation of the edges of $G'_T$ (we remind the reader that $G_T = G^{\mathsf{rev}}$), we construct paths $\{Q_1^T, \ldots, Q_k^T\}$ *reaching* $T$ in $G$ and, by renaming the paths if needed, we assume that, for $i \in [k]$, each $Q_i^S$ is a path starting in $s_i$ and each $Q_i^T$ is a path ending in $t_i$. Moreover, by condition **(1c)** in the definition of R-paths (see Definition 14), each $\mathsf{s}^+(Q_i^S)$ is associated with a unique bag $A_i \in \mathcal{B}$ and each $\mathsf{s}^-(Q_i^T)$ is associated with a unique bag $A'_i \in \mathcal{B}$, such that all bags $A_1, \ldots, A_k, A'_1, \ldots, A'_k$ are distinct. Hence, since $\mathcal{B}$ is a bramble, it follows that for every $i \in [k]$ we can find a shortest path $Q_i$ from $\mathsf{s}^+(Q_i^S)$ to $\mathsf{s}^-(Q_i^T)$ in the strong digraph $G[A_i \cup A'_i]$. Finally, we construct the desired paths $\{P_1, \ldots, P_k\}$, such that each $P_i$ with $i \in [k]$ is a path from $s_i$ to $t_i$, by appending $Q_i$ to $Q_i^S$ and then $Q_i^T$ to the resulting path. Notice that this construction may result in a walk instead of a path, but every walk can be easily shortened into a path $P_i$.

We now claim that every vertex of $G$ appears in at most $c$ paths of the collection $\{P_1, \ldots, P_k\}$. First, notice that since the paths $\{Q_1^S, \ldots, Q_k^S\}$ are disjoint and the paths $\{Q_1^T, \ldots, Q_k^T\}$ are disjoint as well, any vertex not appearing in any bag of $\mathcal{B}$ can appear in at most two paths of $\{P_1, \ldots, P_k\}$. Assume now that $v$ is a vertex appearing in some bag of $\mathcal{B}$. Depending on where $v$ is located in the paths $Q_i^S$, $Q_i^T$, and $Q_i$, we associate $v$ with a bag of $\mathcal{B}$. Since $\mathcal{B}$ has congestion $c$, this immediately validates the claim and the result follows. We remind the reader of our assumption that $\mathscr{P}$ is $\mathcal{B}$-minimal, and look again at Equation 1. If $v$ is in $V_S$ because $v$ is in path $P \in \mathscr{P}^S \cup \mathscr{P}^T$ and $v \neq \mathsf{s}^+(P)$, then we say that $v$ is a *type* 1 vertex. Otherwise, we say that $v$ is a *type* 2 vertex.

For $i \in [k]$, if $v$ is a internal vertex of some $Q_i^S$, then $v \in V^S$ and is either a type 1 or a type 2 vertex. If $v$ is of type 1, then $v$ is an internal vertex of some path $P \in \mathscr{P}^S$ and $v \neq \mathsf{s}^+(P)$. Since $\mathcal{P}$ is $\mathcal{B}$-minimal, this implies that $v$ is in some destination bag of $\mathcal{B}_S$ and we associate $v$ with this bag. If $v$ is of type 2, then $v$ is not an internal vertex of any path in $\mathscr{P}^S$ and is in some bag of $\mathcal{B}_S$. We associate $v$ with this bag. Since the paths $\{Q_1^S, \ldots, Q_k^S\}$ are disjoint, $v$ appears only in one of those paths and thus no other $Q_j^S$ can associate $v$ with another bag of $\mathcal{B}$. The analysis is similar if $v$ is an internal vertex of some $Q_j^T$. Notice that it is possible that $v$ is in both $Q_i^S$ and $Q_j^T$ and, in this case, those two paths associate $v$ with two distinct bags of $\mathcal{B}_S$ and $\mathcal{B}_T$, respectively.

Now let $\mathcal{B}^2 \subseteq \mathcal{B}'$ and let $h' : \mathscr{P} \to \mathcal{B}^2$ be a bijective mapping as in **(1c)** of Definition 14. If $v$ is a vertex of some $P_i$ from $\mathsf{s}^+(Q_i^S)$ to $\mathsf{s}^-(Q_i^T)$ then we associate $v$ with $h'(Q_i^S)$ if $v \in h'(Q_i^S)$, and we associated $v$ with $h'(Q_i^T)$ if $v \in h'(Q_i^T)$.

Now, for $i, j \in [k]$, every path of the form $Q_i^S$, $Q_i^T$, or $P_i$ associates each of its vertex inside of the bramble with a unique bag of $\mathcal{B}$, each vertex associated with some bag appears in $V(Q_i^S) \setminus \{\mathsf{s}^+(Q_i^S)\}$, $V(Q_i^T) \setminus \{\mathsf{s}^-(Q_i^T)\}$, or $V(Q_i)$, no two distinct $Q_i^S, Q_j^T$ associate a vertex $v$ with the same bag, and the same holds with relation to distinct pairs of paths of the form $Q_i, Q_j$ and $Q_i^T, Q_j^T$. We remark that while it is possible that some $v$ appears in both $Q_i^S$ and $Q_i^T$, this does not pose an issue since in this case $v$ is associated with a pair of distinct bags $B \in \mathcal{B}_S$ and $B' \in \mathcal{B}_T$ by $Q_i^S$ and $Q_i^T$, respectively. Since $\mathcal{B}$ has congestion $c$, it follows that every vertex is associated with at most $c$ bags, which implies that every vertex is in at most $c$ paths of $\{P_1, \ldots, P_k\}$, and the result follows.

The bound on the running time follows by Theorem 15 and by observing that a set of R-paths can be made $\mathcal{B}$-minimal in time $\mathcal{O}(c \cdot k^2 \cdot n^2)$ ◀

**Application to the asymmetric version of $(k, c)$-DDP.** Theorem 16 is a direct translation of Giannopoulou et al. [10, Theorem 9.1] to our setting. As mentioned in the introduction, we can prove a weaker version of Theorem 17 by replacing Theorem 16 by [10, Theorem 9.1].

In comparison with the result by Edwards et al. [8], with this approach we can drop the bound on the strong connectivity of the digraph from $(36k^3 + 2k)$ to $k$, and the trade-off is that in this case we have to rely on the topology of *cylindrical grids* to connect the paths, instead of a bramble of congestion $c$. Although it is true that every digraph with large directed tree-width contains a large cylindrical grid [14], and that such a grid can be found in FPT time [4], to find a cylindrical grid of order $k$ the directed tree-width of the digraph has to increase much more than it is needed to find a bramble of congestion two (although both dependencies still consist of a non-elementary tower of exponentials). Additionally, we remark that if the goal is to solve the $(k, c)$-DDP problem with $c \geq 8$, then as stated in Theorem 17 a bramble of congestion eight suffices, and a polynomial dependency on how large the directed tree-width of a digraph must be to guarantee the existence of a large bramble of congestion eight was shown by Masarík et al. [17].

On the other hand, Theorem 16 improves upon [10, Theorem 9.1] in both its statement, since we use brambles instead of cylindrical grids, and in simplicity. Indeed, brambles of bounded congestion seem to be a weaker structure than cylindrical grids, since it possible to extract such brambles with order $t$ from a cylindrical grid of order at least $2t$ (see [8, Lemma 9]), and the bound on how large the directed tree-width of a digraph has to be to guarantee the existence of such a bramble with size $t$ is, in many cases (see [17] for instance) and as far as we know also in the general case, substantially better than what is needed to find a cylindrical grid with the same order. Additionally, the algorithm to find cylindrical grids runs in FPT time [4] given a certificate of large directed tree-width and, in contrast, a large bramble of congestion two can be found in polynomial time when such certificates are provided, as stated in Proposition 7. Finally, we only ask the bramble to have order $2k(c \cdot k - c + 2) + c(k - 1)$ (which equals $4k^2 + 2k - 2$ when $c = 2$) instead of order $k(6k^2 + 2k + 3)$ for the cylindrical grid in [10, Theorem 9.1], where the goal is to compute solutions for $(k, 2)$-DDP. Their proof relies on the topology of cylindrical grids to connect the paths inside of this structure, after some careful selection on how to reach it from $S$ and leave it to reach $T$. In our proof of Theorem 16, it is very simple to connect the paths inside the bramble. Indeed, after applying twice the duality between R-paths and R-cuts, for each $i \in [k]$ we simply connect the ending vertex of the path from $s_i$ to the bramble containing the starting vertex of the path from the bramble to $t_i$, using the strong connectivity of the digraph induced by $B \cup B'$, where $B$ is the bag associated with $s_i$ and $B'$ the bag associated with $t_i$.

Their result [10, Theorem 9.1] is one of the cornerstones in their algorithm to solve the asymmetric version of $(k, 2)$-DDP. Recall that, given ordered sets of terminals $\{s_1, \ldots, s_k\}$ and $\{t_1, \ldots, t_k\}$, the goal in this asymmetric version is to either produce a collection of paths from each $s_i$ to the corresponding $t_i$ such that every vertex is in at most two paths of the collection, or conclude that there is no collection of disjoint $\{s_i\} \to \{t_i\}$ paths. In the first case, we say that we have constructed a *half-integral linkage*. In the second case, we say that we have a no-instance. At any point of their dynamic programming algorithm, if one of the subproblems they define deals with an instance in which there is no small separator intersecting all paths from $S$ to the grid or from the grid to $T$, then they apply [10, Theorem 9.1] to find a solution to the instance. If a separator is found, then they generate two easier instances, one of bounded directed tree-width, and one with fewer number of terminals. Intuitively, the same holds true if we substitute [10, Theorem 9.1] by our Theorem 16. In the full version of this paper, we give an informal sketch of why this is indeed the case.

## References


1 Saeed Akhoondian Amiri, Stephan Kreutzer, Dániel Marx, and Roman Rabinovich. Routing with congestion in acyclic digraphs. *Information Processing Letters*, 151:105836, 2019. doi:doi.org/10.1016/j.ipl.2019.105836.

2 Saeed Akhoondian Amiri, Stephan Kreutzer, Dániel Marx, and Roman Rabinovich. Routing with congestion in acyclic digraphs. *Information Processing Letters*, 151, 2019. doi:10.1016/j.ipl.2019.105836.

3 Jørgen Bang-Jensen and Gregory Gutin. *Classes of Directed Graphs*. Springer Monographs in Mathematics, 2018. doi:10.1007/978-3-319-71840-8.

4 Victor Campos, Raul Lopes, Ana Karolinna Maia, and Ignasi Sau. Adapting the Directed Grid Theorem into an FPT Algorithm. *SIAM Journal on Discrete Mathematics*, 36(3):1887–1917, 2022. doi:10.1137/21M1452664.

5 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, and Dániel Marx. *Parameterized Algorithms*. Springer, 2015.

6 Matthias Dehmer and Frank Emmert-Streib. *Quantitative Graph Theory: Mathematical Foundations and Applications*. Discrete Mathematics and its Applications. Chapman and Hall/CRC, 2014.

7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

8 Katherine Edwards, Irene Muzi, and Paul Wollan. Half-Integral Linkages in Highly Connected Directed Graphs. In *Proc. of the 25th Annual European Symposium on Algorithms (ESA)*, volume 87 of *LIPIcs*, pages 36:1–36:12, 2017. Full version available in arXiv:1611.01004. doi:10.4230/LIPIcs.ESA.2017.36.

9 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

10 Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-joung Kwon. The canonical directed tree decomposition and its applications to the directed disjoint paths problem, 2020. This article is the full version of reference [11]. arXiv:2009.13184.

11 Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-joung Kwon. Directed tangle tree-decompositions and applications. In *Proc. of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–405, 2022. The full version of this article is reference [10]. doi:10.1137/1.9781611977073.19.

12 Thor Johnson, Neil Robertson, Paul Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(01):138–154, 2001. doi:10.1006/jctb.2000.2031.

13 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Stephan Kreutzer. An excluded half-integral grid theorem for digraphs and the directed disjoint paths problem. In *Proc. of the 46th Annual ACM on Symposium on Theory of Computing (STOC)*, pages 70–78, 2014. doi:10.1145/2591796.2591876.

14 Ken-ichi Kawarabayashi and Stephan Kreutzer. The Directed Grid Theorem. In *Proc. of the 47th Annual ACM on Symposium on Theory of Computing (STOC)*, pages 655–664. ACM, 2015. doi:10.1145/2746539.2746586.

15 Dénes Kőnig. Gráfok és mátrixok. *Matematikai és Fizikai Lapok*, 38:116–119, 1931.

16 Raul Lopes and Ignasi Sau. A relaxation of the Directed Disjoint Paths problem: A global congestion metric helps. *Theoretical Computer Science*, 898:75–91, 2022. doi:10.1016/j.tcs.2021.10.023.

17 Tomás Masarík, Marcin Pilipczuk, Pawel Rzazewski, and Manuel Sorge. Constant congestion brambles in directed graphs. *SIAM Journal on Discrete Mathematics*, 36(2):922–938, 2022. doi:10.1137/21m1417661.

18 Karl Menger. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927. URL: http://eudml.org/doc/211191.

19 Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, 2010. doi:10.1137/070697781.

20 Carsten Thomassen. Highly connected non-2-linked digraphs. *Combinatorica*, 11(4):393–395, 1991. doi:10.1007/BF01275674.

# Enumerating Maximal Induced Subgraphs

**Yixin Cao**
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

## Abstract

Given a graph $G$, the maximal induced subgraphs problem asks to enumerate all maximal induced subgraphs of $G$ that belong to a certain hereditary graph class. While its optimization version, known as the minimum vertex deletion problem in literature, has been intensively studied, enumeration algorithms were only known for a few simple graph classes, e.g., independent sets, cliques, and forests, until very recently [Conte and Uno, STOC 2019]. There is also a connected variation of this problem, where one is concerned with only those induced subgraphs that are connected. We introduce two new approaches, which enable us to develop algorithms that solve both variations for a number of important graph classes. A general technique that has been proven very powerful in enumeration algorithms is to build a solution map, i.e., a multiple digraph on all the solutions of the problem, and the key of this approach is to make the solution map strongly connected, so that a simple traversal of the solution map solves the problem. First, we introduce retaliation-free paths to certify strong connectedness of the solution map we build. Second, generalizing the idea of Cohen, Kimelfeld, and Sagiv [JCSS 2008], we introduce an apparently very restricted version of the maximal (connected) induced subgraphs problem, and show that it is equivalent to the original problem in terms of solvability in incremental polynomial time. Moreover, we give reductions between the two variations, so that it suffices to solve one of the variations for each class we study. Our work also leads to direct and simpler proofs of several important known results.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** enumeration algorithm, hereditary graph class, maximal induced subgraph

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.31

**Related Version** *Full Version*: `http://arxiv.org/abs/2004.09885`

**Funding** Supported in part by the Hong Kong Research Grants Council (RGC) under grant 15221420 and the National Natural Science Foundation of China (NSFC) under grant 61972330.

## 1 Introduction

A vertex deletion problem asks to transform a given graph into a graph in a certain graph class $\mathcal{P}$ by deleting vertices. Many classic optimization problems belong to the family of vertex deletion problems, and their algorithms and complexity have been intensively studied. More often than not, the graph class is *hereditary*, i.e., closed under taking induced subgraphs. Examples include complete graphs, edgeless graphs, acyclic graphs, bipartite graphs, planar graphs, and perfect graphs. For a hereditary graph class, this problem is either NP-hard or trivial [40]. An equivalent formulation of a vertex deletion problem toward $\mathcal{P}$ is to ask for a maximum induced subgraph that belongs to $\mathcal{P}$. A plethora of algorithms, including approximation [42, 14, 12, 51, 36, 1], exact [30, 29], and parameterized [10, 15, 14, 12, 36, 1, 2], have been proposed for both formulations.

Yet another approach toward this problem is to enumerate, or generate or list, inclusion-wise *maximal* induced subgraphs that belong to the graph class $\mathcal{P}$, hence called the *maximal induced $\mathcal{P}$ subgraphs* problem. Trivially, one can always use an enumeration algorithm to solve the optimization version of the same problem. A classical example of nontrivial use is to color a graph by enumerating its maximal independent sets [38, 27, 9, 6]. Indeed, the enumeration of maximal independent sets and the enumeration of maximal cliques are practically the same, and both are well-studied classic problems [3, 17, 43]. They were also used in finding


connected vertex covers and edge dominating sets [28]. In general, the maximal induced $\mathcal{P}$ subgraphs problem is motivated by the intrinsic difficulty in formulating a combinatorial optimization problem: there are always factors that are ill characterized or even omitted in the formulation [26, 44].[1] This is particularly the case for vertex deletion problems, of which the primary applications are the processing of noisy data, where modifications to a graph are meant to exclude outliers or to fix noise.

Motivations of maximal induced $\mathcal{P}$ subgraphs problems also come from database theory [18], where one is usually concerned with only induced $\mathcal{P}$ subgraphs that are connected. Of any hereditary graph class $\mathcal{P}$, we can define a subclass by allowing only connected graphs in $\mathcal{P}$. With very few exceptions, this naturally defined class is not hereditary, and hence this variation, called the *maximal connected induced $\mathcal{P}$ subgraphs* problem, poses different challenges. Indeed, as we will see, it is somewhat more difficult than the original one.

Let $n$ denote the number of vertices in the input graph, and by a solution we mean a maximal vertex set that induces a subgraph in $\mathcal{P}$. Since there might be an exponential number (on $n$) of *solutions*, care needs to be taken when we talk about the running time of an enumeration algorithm. For example, in a graph consisting of disjoint triangles, there are $3^{n/3}$ maximal independent sets. Johnson et al. [37] defined three complexity classes for enumeration algorithms, namely, polynomial total time (polynomial in $n$ and the total number of solutions), incremental polynomial time (for all $s$, the time to output the first $s$ solutions is polynomial in $n$ and $s$), and polynomial delay (the time to output the first solution and the time between two consecutive solutions are both polynomial in $n$). Both maximal independent sets and maximal cliques can be enumerated with polynomial delay, and so are maximal bicliques (complete bipartite graphs) [23, 33] and maximal forests [45]. See the survey [48] and the recent results [21, 19].

## 2 Our contributions

Our algorithms are summarized below.

▶ **Theorem 1.** *The maximal induced $\mathcal{P}$ subgraphs problem and its connected variation can be solved with polynomial delay when $\mathcal{P}$ is one of the following graph classes: interval graphs, trivially perfect graphs, split graphs, complete split graphs, pseudo-split graphs, threshold graphs, cluster graphs, complete bipartite graphs, complete $p$-partite graphs, and $d$-degree-bounded graphs.*

▶ **Theorem 2.** *The maximal induced $\mathcal{P}$ subgraphs problem and its connected variation can be solved in incremental polynomial time when $\mathcal{P}$ is one of the following graph classes: wheel-free graphs, unit interval graphs, block graphs, 3-leaf powers, basic 4-leaf powers, and any graph class that can be characterized by a finite set of forbidden induced subgraphs.*

Since a connected cluster graph is a clique, the result for the maximal connected induced cluster subgraphs problem is already known. Also, for a graph class that can be characterized by a finite set of forbidden induced subgraphs, algorithms for the maximal induced $\mathcal{P}$ subgraphs problem, but not its connected variation, can be derived from Eiter and Gottlob [24] (see the discussion in Section 2.6).

[1] After all, how many times do you click "I'm Feeling Lucky" on Google.com?

## 2.1 Solution maps and retaliation-free paths

In a seminal work, Schwikowski and Speckenmeyer [45] proposed an algorithm for enumerating maximal induced forests, as well as its directed variation, enumerating maximal induced directed acyclic subgraphs. They introduced a successor function, which, given a solution, returns a nonempty multi-set of other solutions of $G$. In so doing they *implicitly* built a multiple digraph $\mathcal{M}(G)$ whose nodes are the solutions, and there are arc(s) from node $S_1$ to node $S_2$ if $S_2$ is one of the successors of $S_1$. The key properties of $\mathcal{M}(G)$ are (1) the successor function can be calculated in polynomial time; and (2) $\mathcal{M}(G)$ is strongly connected. As a result, traversing $\mathcal{M}(G)$ from any node, we can visit all the solutions in time polynomial in $n$ and linear on the number of solutions. With a standard bookkeeping mechanism, we can easily implement it with polynomial delay. We call the multiple digraph on the solutions of an enumeration problem its *solution map*. Gély et al. [33] later rediscovered this idea, and used it to re-analyze enumeration algorithms for maximal cliques and maximal bicliques. Solution maps are actually very general and powerful. Conte et al. [21, 19] built solution maps to solve the maximal (connected) induced $\mathcal{P}$ subgraphs problem for several important graph classes. In particular, they solved the connected variation for forests, and directed acyclic graphs. (They also designed algorithms for enumerating maximal subgraphs, a direction that we will not pursue in the present paper, and other non-graphic problems.)

To solve an enumeration problem with a solution map consists in defining the successor function and proving that the implied solution map $\mathcal{M}$ is strongly connected. All the mentioned algorithms follow the same general scheme, although the details are quite problem specific. For convenience, we may assume that the solutions are subsets of some ground set $U$; for the maximal (connected) induced $\mathcal{P}$ subgraphs problem, $U$ is the vertex set of the input graph.

**Successors of a solution $S$:** For each $v \in U \setminus S$, define a sub-instance restricted to $S \cup \{v\}$, and find a set of solutions of this sub-instance. The union of these $|U \setminus S|$ sets makes the successors of $S$.

**Strong connectedness:** For each solution $S^*$, define a specific metric and show that every other solution $S$ has a successor "closer" to $S^*$ than $S$ with respect to this metric.

Since a solution $S'$ can be a successor of $S$ by virtue of two different elements in $U \setminus S$, the solution map is a multiple digraph. The simplest metric is arguably the lexicographical order. If we number the elements such that those in $S^*$ are the smallest, then $S^*$ is the smallest among all solutions, and hence it suffices to show that each solution $S$ has a lexicographically smaller successor. The existence of a path from $S$ to $S^*$ in $\mathcal{M}$, hence the strong connectedness of $\mathcal{M}$, will then follow from the finiteness of the ground set $U$. Other metrics have also been used in literature [45, 18, 21, 19]

The requirement that every other solution has a *direct* successor that is closer to $S^*$, with respect to the metric decided by $S^*$, is not always easy to achieve. For example, such successor functions have been claimed for unit interval graphs and interval graphs, but both turned out to be incorrect. For the connectedness of a solution map, after all, we are only concerned with the reachability: whether a solution can reach $S^*$ instead of how long it takes to do so. (Indeed, in "the most ideal case," the solution map can be a simple directed cycle, in which the distance of a pair of solutions can be the number of solutions minus one.)

We introduce retaliation-free paths to certify strong connectedness of a solution map. Suppose that we are using the lexicographic metric. In our framework, it is possible that all the successors of a solution $S$ are lexicographically larger than $S$. Thus, the strong connectedness of our solution maps is not readily guaranteed, and we proceed as follows. Let $s$ be the smallest element in $S^* \setminus S$. If any successor of $S$ contains $[1..s]$, then it is

lexicographically smaller than $S$ and we are done. Otherwise, we look for a solution $S'$ that contains $[1..s]$ and is reachable from $S_0 = S$ by a nontrivial path $S_0 S_1 \cdots S'$. It is better to view this path as a transforming procedure. In the first step, we choose a successor $S_1$ of $S$ containing $s$; we call $s$ the *gainer*, and elements in $[1..s] \setminus S_1$ the *victims* of $s$. We then try to add the victims of $s$ back, with the guarantee that none of them unseats $s$, an action which we call *retaliation*. A victim $r$ of $s$ may become a gainer in a later step, hence introducing further victims. In this case, we put other victims of $s$ on hold, and try to restore the victims of $r$. Now we need to make sure both $r$ and $s$ are safe. Thus, we consider the gainer–victim relationship transitive. Such a path of solutions in $\mathcal{M}$ is called *retaliation-free*; see, e.g., Figure 1. A retaliation-free path terminates only when it reaches $S^*$. Therefore, it suffices to show that it does terminate. (After we reach a solution containing $[1..s]$, the element $s$ is no longer a gainer, and may be removed as a victim for adding a later element. The path from $S$ to $S^*$ we produced as such can have a length super-polynomial in $n$.)

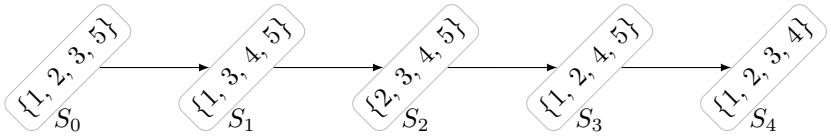
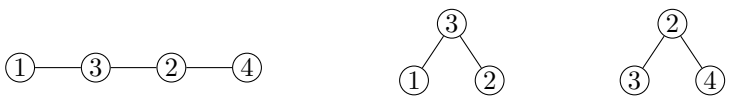



**Figure 1** A path from $S_0$ to $S_4 = S^*$ in a solution map. Here $U = \{1, 2, 3, 4, 5\}$, and its elements are numbered such that those in $S^*$ have the smallest numbers. In the first step ($S_0 \to S_1$), element 4 is the gainer and element 2 is the victim, which becomes the gainer of the second step ($S_1 \to S_2$), with victim 1. The last two steps have no victims.

▶ **Theorem 3** (Informal). *Let $\mathcal{M}$ be the solution map of an enumeration problem, and let $S^*$ be a specific solution. Every retaliation-free path in $\mathcal{M}$ with respect to $S^*$ terminates at $S^*$.*

We use Theorem 3 to develop enumeration algorithms for the maximal (connected) induced trivially perfect subgraphs problem and the maximal (connected) induced interval subgraphs problem, both running with polynomial delay. The two problems are paradigmatic for the use of this technique. The main structures we use for interval graphs are the clique paths, which are linear, while for trivially perfect graphs, we work on their generating forests, which are hierarchical.

A trivially perfect graph $G$ can be represented as a forest $F$, called its generating forest, such that two vertices $u$ and $v$ are adjacent in $G$ if and only if one of them is an ancestor of the other in $F$ [49, 50]. It is simpler if we consider the connected variation, where generating forests are trees. For the purpose of transforming a solution $S$ to another solution $S^*$, we transform a generating tree $T$ of $G[S]$ to a generating tree $T^*$ of $G[S^*]$. A main obstacle is that two vertices in $S \cap S^*$ may have different ancestor–descendant relationship in $T$ and $T^*$; see, e.g., vertices 2 and 3 in Figure 2.

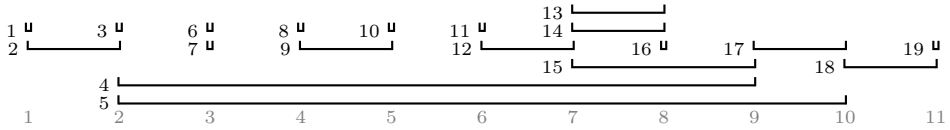



**(a)** $G$. **(b)** $G[S^*] = T^*$. **(c)** $G[S] = T$.

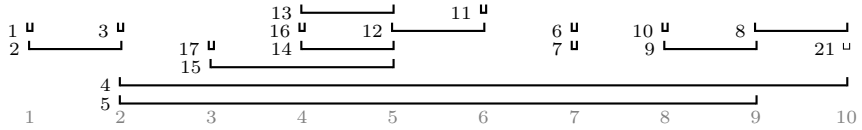
**Figure 2** A graph $G$ and its only two maximal connected induced trivially perfect subgraphs.

We add the vertices in $S^* \setminus S$ bottom-up, i.e., children before parents. Let $v$ be the smallest vertex in $S^* \setminus S$ in the post-order of $T^*$; note that all the $T^*$-descendants of $v$ are in $S$. It is easy to add $v$ if $v$ is a leaf of $T^*$ (because all its $T^*$-ancestors have larger numbers than $v$) or if $v$ has a single child $v'$ in $T^*$ (because $v$ and $v'$ are true twins in $G[S^*]$). The nontrivial case is when $v$ has multiple $T^*$-children. Since we can only make a polynomial number of solutions to be successors of $S$, it is very likely that none of the successors keep all the $T^*$-children of $v$. We observe that as long as we keep two $T^*$-children of $v$, then the relationship between $T^*$-descendants of $v$ and $T^*$-ancestors of $v$ will be correctly maintained. We afford to lose other children of $v$ and their descendants as victims of this step, because they can be added back easily: all the operations for adding these victims back are under $v$, hence isolated from other parts.

Interval graphs are a very important and well-studied graph class [8, 32, 34, 35], and the interval vertex deletion problem receives a lot of attention [7, 14, 11, 2]. The maximal cliques of an interval graph can be arranged in a linear manner, called a clique path [32]. A natural idea is to take a clique path for $G[S^*]$ and number the vertices from left to right; see, e.g., Figure 3(a). Then for another solution $S$, we try to add the smallest vertex in $S^* \setminus S$. However, for solution $S$ shown in Figure 3(b), there is no obvious way to add vertex 18 to $S$, without removing a smaller vertex. The reason is that intervals for vertices 6–17 are organized in very different ways in the two clique paths. For a reader familiar with the recognition of interval graphs, the issue is essentially the same as recognizing interval graphs by graph searches, scanning the vertices from one end to the other end [13]. Note that $\{8, 9, 10\}$ is a *module* (a subset of vertices that have the same neighborhood outside this subset) of $G[S^*]$ and switching the positions of vertices 8 and 10 gives another valid representation of $G[S^*]$. On the other hand, the set $\{6, \ldots, 17\}$ is not a module of $G[S^*]$, and its orientation is fixed from the right by 18. Since this fact can only be deduced after seeing vertex 18, it is not available if one scans from the left. The algorithm in [13] tries to detect and keep track of the orientation of such vertex sets via multiple-sweep searches. For our problem, we allow some vertices in $\{6, \ldots, 17\}$ to fall victims to vertex 18. Afterward, we restore the victims *from right to left*: we process them in a different direction because vertex 18 is the anchor of this set.




**(a)** $S^*$.




**(b)** $S$.

**Figure 3** Two maximal connected induced interval subgraphs of some graph $G$ (other vertices are immaterial and omitted). Gray numbers at the bottom indicate the indices of the maximal cliques.

## 2.2 Algorithms in incremental polynomial time

In each of the mentioned algorithms, the successor function has to be handcrafted. For example, our successor function for interval graphs does not work for unit interval graphs. Nor can the successor function of [21, 19] for chordal graphs be applied to interval graphs.[2] In the following we aim for general approaches that can be applied with little efforts.

There is nothing inherent in solution maps about polynomial delay, though all of the mentioned algorithms based on solution maps are of this type. The running time of such an algorithm is determined by the successor function, which is almost always determined by the number of successors a solution can have. If the number is polynomial in $n$, then the algorithm has polynomial delay. If we allow the number to be polynomial in both $n$ and the number of solutions, then it needs polynomial total time. One "weakness" of the approach based on solution maps is the lack of a simple way to develop algorithms in incremental polynomial time, save those simple classes [18]. We obviate this concern with the following observation.

▶ **Theorem 4.** *For any hereditary graph class $\mathcal{P}$, the maximal (connected) induced $\mathcal{P}$ subgraphs problem can be solved in polynomial total time if and only if it can be solved in incremental polynomial time.*

For the maximal induced $\mathcal{P}$ subgraphs problem, the statement follows from the classical result of Bioch and Ibaraki [5]; see discussions in Section 2.6. As far as we can check, the statement for the connected variation was not previously known. (This explains why Cohen et al. [18] had to prove the statement separately.) Theorem 4 enables us to use solution maps transparently in developing algorithms for enumerating maximal (connected) induced $\mathcal{P}$ subgraphs in incremental polynomial time. There is strong evidence that a similar claim on general enumeration problems does not hold [46, 16].

## 2.3 Restricted versions

Lawler et al. [39] introduced the *input-restricted version* of an enumeration problem: there exists an element $x \in U$ such that $U \setminus \{x\}$ is a solution. This special problem arises naturally in several design paradigms for enumeration problems, e.g., the design of successor functions, and the reverse search technique [4]. Cohen et al. [18] proved that the maximal (connected) induced $\mathcal{P}$ subgraphs problem can be solved with polynomial delay when its input-restricted version can be solved in polynomial time. Moreover, the original problem can be solved in incremental polynomial time if and only if its input-restricted version can. They also proved a similar statement for polynomial total time, which is rendered redundant by Theorem 4. Thus, this ostensibly simpler version is the core of the maximal (connected) induced $\mathcal{P}$ subgraphs problem in terms of solvability in incremental polynomial time.

Since the appearance of [18], there have been attempts at extending its core idea. A "natural" way seems to be defining a restricted version that is equipped with a special set of more than one vertex; i.e., $G - Z$ is in $\mathcal{P}$ for some set $Z$ of $t$ vertices, where $t > 1$. However,

[2] An early version of [19] mistakenly claimed that their algorithm for the maximal induced chordal subgraphs problem also applies to interval graphs. Roughly speaking, what they did is to reconstruct a perfect elimination ordering of a desired solution $S^*$ from a solution $S$. As long as the newly added vertex is simplicial, the resulting graph is chordal. But it is not always an interval graph. The tent graph (adding three edges to connect the even-number vertices of a 6-cycle) is a counterexample. In the solution map they built, the only successor of $V(G) \setminus \{v\}$, where $v$ is any degree-2 vertex of $G$, is $G$ itself, which is not an interval graph. It is quite nontrivial, if possible at all, to adapt this approach to interval graphs.

the extra $t-1$ vertices in the set $Z$ turn out to be not helpful. Consider, for instance, $\mathcal{P}$ being the forests. For a graph $G$ and any $t$, we can make a new graph $H$ by introducing $t-1$ copies of disjoint triangles to $G$, and then $H$ satisfies the new definition if and only if $G$ is input-restricted. Therefore, this special version does not have any property that is not already in the input-restricted version, hence not easier to solve than the latter.

This negative example does not suggest a dead end, and there is a natural generation that does work. We may view the input-restricted version as an enumeration problem by itself, and try to build a solution map to solve it. Then in designing the successor function, in the sub-instance restricted to $S \cup \{v\}$, aside from the vertex $x$ we had, we are bestowed with another vertex $v$ such that the removal of either of $x$ and $v$ leaves a subgraph in $\mathcal{P}$. We are thus inspired to define the *$t$-restricted version* of the maximal induced $\mathcal{P}$ subgraphs problem:

> There exists a set $Z$ of $t$ vertices in $G$ such that $G - z$ is in $\mathcal{P}$ for *every* $z \in Z$.

It is equivalent to that *every* induced subgraph of $G$ that is not in $\mathcal{P}$ contains *all* vertices in $Z$. This requirement is far stronger than $G - Z$ being in $\mathcal{P}$, though equivalent when $t = 1$. We are able to show that even this far more restricted version is still equivalent to the original problem in terms of enumerability in incremental polynomial time. We remark that our proofs, based on solution maps, are significantly simpler than those on the input-restricted version [18], which can now be viewed as the 1-restricted version. With the benefit of hindsight, we can see that all the results of [18] can be easily interpreted using solution maps, with simpler proofs.

▶ **Theorem 5.** *Let $\mathcal{P}$ be a hereditary graph class. The maximal (connected) induced $\mathcal{P}$ subgraphs problem can be solved in incremental polynomial time if and only if there exists a positive integer $t$ such that its $t$-restricted version can be solved in polynomial total time.*

The strong requirement stipulated in defining the $t$-restricted version makes it significantly easier than the original problem. All the algorithms in Theorem 2 are obtained by reducing these problems to their $t$-restricted versions. Of these results, we would like to draw special attention to those on wheel-free graphs, though this graph class in its own sense may seem to be less interesting compared to others we study. Lokshtanov [41] proved that the vertex deletion problem toward wheel-free graphs is W[2]-hard with respect to standard parameterization, hence very unlikely fixed-parameter tractable. This suggests that the complexity of the maximal induced $\mathcal{P}$ subgraphs problem can be quite different from its optimization counterpart.

Theorem 5 also implies, among others, the following result on graph classes that can be characterized by a finite set $\mathcal{F}$ of forbidden induced subgraphs. Indeed, with $t$ being the maximum order of graphs in $\mathcal{F}$, the $t$-restricted version of the maximal (connected) induced $\mathcal{F}$-free subgraphs problem is trivial. Eiter and Gottlob [24] have proved this result for the maximal induced $\mathcal{F}$-free subgraphs problem, but their proof does not apply to the connected variation; see discussions in Section 2.6.

▶ **Corollary 6.** *Let $\mathcal{F}$ be a finite set of graphs. The maximal (connected) induced $\mathcal{F}$-free subgraphs problem can be solved in incremental polynomial time.*

We note that the vertex deletion problem to $\mathcal{F}$-free graphs for finite $\mathcal{F}$ has been well studied. In particular, they are fixed-parameter tractable [10] and admit constant-approximation [42].

## 2.4 Reductions between the two variations

Since we are dealing with both the maximal induced $\mathcal{P}$ subgraphs problem and its connected variation, it is really irksome if we have to develop two algorithms for each class. Although Cohen et al. [18] and Conte et al. [21, 19] dealt with both variations, they stopped at noting that with one of them solved, a slight modification would be able to solve the other. These modifications have to be done, however, case by case.

The main issue of the connected variation is that the set of connected graphs in a hereditary graph class $\mathcal{P}$, viewed as a graph class in its own regard, is mostly not hereditary. If there are two nonadjacent vertices in any connected graph $G$ in $\mathcal{P}$, then the edgeless graph on two vertices is an induced subgraph of $G$, hence in $\mathcal{P}$. Therefore, the class of connected graphs in $\mathcal{P}$ remains hereditary only when $\mathcal{P}$ is the class of cluster graphs, in which the connected ones are the complete graphs, and the class of edgeless graphs. A connected edgeless graph has precisely one vertex, and this seems to be the only class on which the connected variation, which is trivial, is the easier between the two variations. For a hereditary graph class $\mathcal{P}$, and a graph $G$, let us use $N_1$ and $N_2$ to denote, respectively, the number of maximal induced $\mathcal{P}$ subgraphs and the number of maximal connected induced $\mathcal{P}$ subgraphs of $G$. It is not difficult to see that $N_1/N_2$ can be an exponential number on $n$, while $N_2/N_1$ can never be more than $n$. In the trivial case when $\mathcal{P}$ is edgeless, $N_2$ is precisely $n$, while $N_1$ can be $3^{n/3}$. For a nontrivial example, the graph consisting of $n/4$ disjoint 4-cycles has $2^{n/2}$ maximal induced forests, while only $n$ maximal induced trees. This example applies to interval graphs, chordal graphs, and many others. Therefore, it is very unlikely we can use an enumeration algorithm for the maximal induced $\mathcal{P}$ subgraphs problem to solve its connected variation.

The other direction is promising. Two simple observations help here. First, if we can add vertices to a graph without making new forbidden induced subgraphs, then it does not change the number of maximal induced $\mathcal{P}$ subgraphs, though each of them gets more vertices. Second, if the extra vertices make all the maximal induced $\mathcal{P}$ subgraphs connected, then by enumerating all maximal connected induced $\mathcal{P}$ subgraphs of the new graph, we obtain effortlessly all maximal induced $\mathcal{P}$ subgraphs of the original graph. This idea works when $\mathcal{P}$ is closed under adding universal vertices, examples of which include interval graphs, chordal graphs, etc. A less trivial reduction can be devised for several other graph classes, e.g., wheel-free graphs and triangle-free graphs. These reductions focus us on the connected variation of the problems only, instead of working on both.

Via Theorem 5, we are able to establish a more interesting connection. Although the condition in the statement looks rather technical, it is satisfied by many natural graph classes: apart from those in Theorem 2, another notable example is planar graphs. As for the $t$-restricted version of both variations, there are only a polynomial number of solutions that are not a proper superset of $Z$, and they are easy to handle. Therefore, the focus is on those solutions containing all vertices in $Z$. We observe that under the connectivity condition, in every maximal induced $\mathcal{P}$ subgraph of $G$ that contains all vertices in $Z$, all the vertices in $Z$ are always in the same component. As a result, there is a one-to-one mapping between such solutions for these two variations.

▶ **Theorem 7.** *Let $c$ be a constant. Let $\mathcal{F}$ be a set of graphs such that every graph in $\mathcal{F}$ of order $c$ or above is biconnected. The maximal induced $\mathcal{F}$-free subgraphs problem can be solved in incremental polynomial time if and only if its connected variation can be solved in incremental polynomial time.*

## 2.5 Characterizing the easy classes

We try to better understand maximal (connected) induced $\mathcal{P}$ subgraphs problems that can be solved with polynomial delay by considering its input-restricted version. We say that a graph class $\mathcal{P}$ has the *CKS property*, after the initials of the authors of [18], if the input-restricted version of the maximal (connected) induced $\mathcal{P}$ subgraphs problem can be solved in time polynomial only in $n$. Since it is straightforward to see that the class of edgeless graphs (independent sets), the class of complete graphs (cliques), and the class of complete bipartite graphs (bicliques) have the CKS property, the polynomial-delay algorithms for them can be immediately explained by this general result. On the other hand, the polynomial-delay algorithms of [45] and [21, 19], and our polynomial-delay algorithms for the maximal (connected) induced interval subgraphs problem and for the maximal (connected) induced trivially perfect subgraphs problem cannot be derived from this observation. As we will see, none of these graph classes has the CKS property.

It turns out that star forests, forests in which every tree is a star, play a crucial role in characterizing graph classes with the CKS property. Therefore, to find those graph classes with the CKS property, it suffices to consider those forbidding both a star forest and the complement of a star forest.

▶ **Theorem 8.** *Let $\mathcal{F}$ be a nonempty set of graphs. If the class of $\mathcal{F}$-free graphs has the CKS property, then $\mathcal{F}$ contains at least one star forest and the complement of at least one star forest.*

Unfortunately, this condition is not sufficient, and many graph classes satisfy this condition but do not have the CKS property. Moreover, it is possible that a class $\mathcal{P}$ of graphs has the CKS property but a proper subclass of $\mathcal{P}$ does not. This fact makes a full characterization of graph classes with the CKS property more difficult.

## 2.6 Related work

Let us put our work into context. The aforementioned characterization of a hereditary graph class by a set $\mathcal{F}$ of forbidden induced subgraphs provides another perspective to view the maximal induced $\mathcal{P}$ subgraphs problem, but not its connected variation in general. A subset $S$ of vertices is a solution if and only if $V(G) \setminus S$ intersects all *forbidden sets* of $G$, i.e., a minimal set $X$ of vertices such that $G[X] \in \mathcal{F}$. If we list all the forbidden sets of $G$ as a set system, then the problem becomes enumerating *minimal* vertex sets that intersect each of the forbidden sets. This brings us to the well-studied problem of enumerating minimal hitting sets of a set system. A set system is also known as a *hypergraph*, and the problem is called *hypergraph transversal* in literature. Some important general results on enumeration of maximal induced $\mathcal{P}$ subgraphs, e.g., the algorithm for graph classes characterized by a finite number of forbidden induced subgraphs, were first proved in this setting [24]. Reducing to the hypergraph transversal problem was also a common approach used by many earlier heuristic enumeration algorithms, for maximal independent sets and for maximal forests. A graph may have an exponential number of simple cycles, and thus the maximal induced forests problem and its associated hypergraph transversal problem have significantly different input sizes: the latter, including the number of elements and the number of forbidden sets, may be exponential on the former. This can happen for all graph classes $\mathcal{P}$ that have an infinite number of forbidden induced subgraphs. Deciding whether a graph belongs to such a class $\mathcal{P}$ may be NP-hard, and hence the maximal induced $\mathcal{P}$ subgraphs problem cannot be solved in polynomial total time, though it is still possible for the corresponding hypergraph

transversal problem, with all the forbidden sets given. Thus, we have to exclude from our study those graph classes that cannot be recognized in polynomial time. For a graph class that can be recognized in polynomial time, the maximal (connected) induced $\mathcal{P}$ subgraphs problem is in EnumP, the counterpart of NP for enumeration [16]. (One may consider the oracle model, where whether a graph is in the class is answered by an oracle in $O(1)$ time, but this paper will not take this direction.)

The (minimal) hitting sets of a hypergraph $H$ define another hypergraph $H'$, and it is an easy exercise to verify that each set in $H$ is a (minimal) hitting set of $H'$. The hypergraph transversal problem has a decision version: given a pair of hypergraphs on the same set of elements, decide whether one consists of exactly the minimal hitting sets of the other. Bioch and Ibaraki [5] showed that the hypergraph traversal problem can be solved in polynomial total time if and only if it can be solved in incremental polynomial time, by showing that this is the case precisely if the decision version can be solved in polynomial time. Note that under certain complexity assumptions, these two complexity classes for enumeration problems are not equal in general [46, 16].

The results of [5] were actually developed in the setting of monotone Boolean functions. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $x \leq y$ always implies $f(x) \leq f(y)$. The *identification* problem is to find all minimal true vectors *and* all maximal false vectors of $f$. It is easy to see that the maximal induced $\mathcal{P}$ subgraphs problem is a special case of it. Let $G$ be a graph on $n$ vertices, for any subset $X \subseteq V(G)$, we use $x$ to denote the characteristic vector of $X$, i.e., an $n$-dimension Boolean vector with 1 at the $i$th position if and only if $v_i \in X$, and set $f(x) = 0$ if and only if $G[X] \in \mathcal{P}$. Then minimal forbidden sets and the vertex sets of maximal induced $\mathcal{P}$ subgraphs of $G$ correspond to minimal true vectors and maximal false vectors of $f$, respectively. In this terminology, the maximal induced $\mathcal{P}$ subgraphs problem asks for maximal false vectors only, hence different in the output size. (Moreover, the identification problem is usually asked in the oracle model.) The decision version of the identification problem, known as dualization of monotone Boolean functions, is equivalent to the hypergraph traversal problem, and its incarnations can be found in database systems, artificial intelligence, and game theory. Fredman and Khachiyan [31] proved that dualization of monotone Boolean functions, hence hypergraph traversal, can be solved in quasi-polynomial time. It has been open for nearly four decades whether this problem can be solved in polynomial time; see the survey of Eiter et al. [25].

The main motivation of studying the connected variation is its practical applications in database theory, where several important problems can be modeled as enumerating maximal connected induced $\mathcal{P}$ subgraphs. We refer to Cohen et al. [18] and references therein for the background. Since this variation cannot be directly cast into hypergraphs or monotone Boolean functions, fewer results on it have been known in literature [18, 21, 19], and they had to be dealt with separately.

We are exclusively concerned with maximal (connected) induced $\mathcal{P}$ subgraphs. There is also research on enumerating all induced $\mathcal{P}$ subgraphs and enumerating all maximum induced $\mathcal{P}$ subgraphs. The first is usually very easy, if not completely trivial, while the latter has to be very hard: after all, finding a single maximum induced subgraph in a nontrivial and hereditary graph class is already NP-hard [40]. Yet another, probably more practical, approach is to list top $k$ solutions, or solutions of costs at most (or at least) $k$. This is frequently studied in the framework of parameterized computation. There is work using the input size as the sole measure for the running time of enumeration algorithms, e.g., [47]. In summary, efforts toward a systematic understanding of enumeration are gaining momentum [22].

Our final remark is on the space usage. A naïve implementation of our main approaches needs to keep track of all the solutions, hence takes exponential space. How to reduce the space is the major open problem in this area, and positive results are few but growing. Using reverse search, Conte et al. [20] showed how to strengthen polynomial-delay algorithms of [18] to use only polynomial space. Conte et al. [21, 19] also managed to reduce some of their algorithms to polynomial space.

## References


1. Akanksha Agrawal, Daniel Lokshtanov, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Feedback vertex set inspired kernel for chordal vertex deletion. *ACM Transactions on Algorithms*, 15(1):11:1–11:28, 2019. doi:10.1145/3284356.
2. Akanksha Agrawal, Pranabendu Misra, Saket Saurabh, and Meirav Zehavi. Interval vertex deletion admits a polynomial kernel. In Timothy M. Chan, editor, *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1711–1730. SIAM, 2019. doi:10.1137/1.9781611975482.103.
3. E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2(1):1–6, 1973. doi:10.1137/0202001.
4. David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996. doi:10.1016/0166-218X(95)00026-N.
5. Jan C. Bioch and Toshihide Ibaraki. Complexity of identification and dualization of positive boolean functions. *Information and Computation*, 123(1):50–63, 1995. doi:10.1006/inco.1995.1157.
6. Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. doi:10.1137/070683933.
7. Ivan Bliznets, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Largest chordal and interval subgraphs faster than $2^n$. *Algorithmica*, 76(2):569–594, 2016. doi:10.1007/s00453-015-0054-2.
8. Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.
9. Jesper Makholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, 2004. doi:10.1016/j.orl.2004.03.002.
10. Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.
11. Yixin Cao. Linear recognition of almost interval graphs. In Robert Krauthgamer, editor, *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1096–1115. SIAM, 2016. Full version available at arXiv:1403.1515. doi:10.1137/1.9781611974331.ch77.
12. Yixin Cao. Unit interval editing is fixed-parameter tractable. *Information and Computation*, 253:109–126, 2017. doi:10.1016/j.ic.2017.01.008.
13. Yixin Cao. Recognizing (unit) interval graphs by zigzag graph searches. In Hung Viet Le and Valerie King, editors, *Proceedings of the 4th SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 92–106. SIAM, 2021. doi:10.1137/1.9781611976496.11.
14. Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms*, 11(3):21:1–21:35, 2015. doi:10.1145/2629595.
15. Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016. doi:10.1007/s00453-015-0014-x.
16. Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discrete Applied Mathematics*, 268:179–190, 2019. doi:10.1016/j.dam.2018.06.038.

17 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.

18 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147–1159, 2008. doi:10.1016/j.jcss.2008.04.003.

19 Alessio Conte, Roberto Grossi, Andrea Marino, Takeaki Uno, and Luca Versari. Proximity search for maximal subgraph enumeration. *SIAM Journal on Computing*, 51(5):1580–1625, 2022. doi:10.1137/20m1375048.

20 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Listing maximal subgraphs satisfying strongly accessible properties. *SIAM Journal on Discrete Mathematics*, 33(2):587–613, 2019. doi:10.1137/17M1152206.

21 Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 1179–1190. ACM, 2019. doi:10.1145/3313276.3316402.

22 Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. A complexity theory for hard enumeration problems. *Discrete Applied Mathematics*, 268:191–209, 2019. doi:10.1016/j.dam.2019.02.025.

23 Vânia M. F. Dias, Celina M. H. de Figueiredo, and Jayme Luiz Szwarcfiter. Generating bicliques of a graph in lexicographic order. *Theoretical Computer Science*, 337(1-3):240–248, 2005. doi:10.1016/j.tcs.2005.01.014.

24 Thomas Eiter and Georg Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995. doi:10.1137/S0097539793250299.

25 Thomas Eiter, Kazuhisa Makino, and Georg Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008. doi:10.1016/j.dam.2007.04.017.

26 David Eppstein. Finding the $k$ shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

27 David Eppstein. Small maximal independent sets and faster exact graph coloring. *J. Graph Algorithms Appl.*, 7(2):131–140, 2003. doi:10.7155/jgaa.00064.

28 Henning Fernau. Edge dominating set: Efficient enumeration-based exact algorithms. In Hans L. Bodlaender and Michael A. Langston, editors, *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *LNCS*, pages 142–153. Springer, 2006. doi:10.1007/11847250_13.

29 Fedor V. Fomin, Serge Gaspers, Daniel Lokshtanov, and Saket Saurabh. Exact algorithms via monotone local search. *Journal of the ACM*, 66(2):8:1–8:23, 2019. doi:10.1145/3284176.

30 Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and CMSO. *SIAM Journal on Computing*, 44(1):54–87, 2015. doi:10.1137/140964801.

31 Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996. doi:10.1006/jagm.1996.0062.

32 Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965. doi:10.2140/pjm.1965.15.835.

33 Alain Gély, Lhouari Nourine, and Bachir Sadi. Enumeration aspects of maximal cliques and bicliques. *Discrete Applied Mathematics*, 157(7):1447–1459, 2009. doi:10.1016/j.dam.2008.10.010.

34 Wen-Lian Hsu. $O(m{\cdot}n)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM Journal on Computing*, 24(3):411–439, 1995. doi:10.1137/S0097539793260726.

35 Wen-Lian Hsu and Tze-Heng Ma. Fast and simple algorithms for recognizing chordal comparability graphs and interval graphs. *SIAM Journal on Computing*, 28(3):1004–1020, 1999. doi:10.1137/S0097539792224814.

**36** Bart M. P. Jansen and Marcin Pilipczuk. Approximation and kernelization for chordal vertex deletion. In Philip N. Klein, editor, *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1399–1418. SIAM, 2017. doi:10.1137/1.9781611974782.91.

**37** David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.

**38** Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976. doi:10.1016/0020-0190(76)90065-X.

**39** Eugene L. Lawler, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980. doi:10.1137/0209042.

**40** John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

**41** Daniel Lokshtanov. Wheel-free deletion is W[2]-hard. In Martin Grohe and Rolf Niedermeier, editors, *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5018 of *LNCS*, pages 141–147, New York, 2008. Springer. doi:10.1007/978-3-540-79723-4_14.

**42** Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. In Andrzej Lingas, Rolf G. Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming (ICALP)*, volume 700 of *LNCS*, pages 40–51. Springer, 1993. doi:10.1007/3-540-56939-1_60.

**43** Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In Torben Hagerup and Jyrki Katajainen, editors, *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory, SWAT 2004*, volume 3111 of *LNCS*, pages 260–272. Springer, 2004. doi:10.1007/978-3-540-27810-8_23.

**44** Andrea Marino. *Analysis and Enumeration: Algorithms for Biological Graphs*, volume 6 of *Atlantis Studies in Computing*. Atlantis, 2015. doi:10.2991/978-94-6239-097-3.

**45** Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(1-3):253–265, 2002. doi:10.1016/S0166-218X(00)00339-5.

**46** Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot – Paris 7, Paris, France, 2010.

**47** Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006. doi:10.1016/j.tcs.2006.06.015.

**48** Kunihiro Wasa. Enumeration of enumeration algorithms. arXiv:1605.05102, 2016.

**49** E. S. Wolk. The comparability graph of a tree. *Proceedings of the American Mathematical Society*, 13:789–795, 1962. doi:10.1090/S0002-9939-1962-0172273-0.

**50** Jing-Ho Yan, Jer-Jeong Chen, and Gerard Jennhwa Chang. Quasi-threshold graphs. *Discrete Applied Mathematics*, 69(3):247–255, 1996. doi:10.1016/0166-218X(96)00094-7.

**51** Jie You, Jianxin Wang, and Yixin Cao. Approximate association via dissociation. *Discrete Applied Mathematics*, 219:202–209, 2017. doi:10.1016/j.dam.2016.11.007.

# Approximation Algorithm for Norm Multiway Cut

**Charlie Carlson**
University of Colorado Boulder, CO, USA

**Jafar Jafarov**
Toyota Technological Institute at Chicago, IL, USA

**Konstantin Makarychev**
Northwestern University, Evanston, IL, USA

**Yury Makarychev**
Toyota Technological Institute at Chicago, IL, USA

**Liren Shan**
Northwestern University, Evanston, IL, USA

## Abstract

We consider variants of the classic Multiway Cut problem. Multiway Cut asks to partition a graph $G$ into $k$ parts so as to separate $k$ given terminals. Recently, Chandrasekaran and Wang (ESA 2021) introduced $\ell_p$-norm Multiway Cut, a generalization of the problem, in which the goal is to minimize the $\ell_p$ norm of the edge boundaries of $k$ parts. We provide an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation algorithm for this problem, improving upon the approximation guarantee of $O(\log^{3/2} n \log^{1/2} k)$ due to Chandrasekaran and Wang.

We also introduce and study Norm Multiway Cut, a further generalization of Multiway Cut. We assume that we are given access to an oracle, which answers certain queries about the norm. We present an $O(\log^{1/2} n \log^{7/2} k)$ approximation algorithm with a weaker oracle and an $O(\log^{1/2} n \log^{5/2} k)$ approximation algorithm with a stronger oracle. Additionally, we show that without any oracle access, there is no $n^{1/4-\varepsilon}$ approximation algorithm for every $\varepsilon > 0$ assuming the Hypergraph Dense-vs-Random Conjecture.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Theory of computation → Facility location and clustering

**Keywords and phrases** Multiway cut, Approximation algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.32

**Related Version** *Full Version*: https://arxiv.org/abs/2308.08373

**Funding** *Konstantin Makarychev*: supported by NSF Awards CCF-1955351, CCF-1934931, EECS-2216970.
*Yury Makarychev*: supported by NSF CCF-1955173, CCF-1934843, and ECCS-2216899.
*Liren Shan*: supported by NSF Awards CCF-1955351, CCF-1934931, and EECS-2216970.

## 1 Introduction

In this paper, we consider a variant of the classic combinatorial optimization problem, Minimum Multiway Cut. Given an undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$ and $k$ terminals $t_1, \ldots, t_k \in V$, the Minimum Multiway Cut problem asks to partition graph $G$ into $k$ parts $P_1, \ldots, P_k$ so that $P_i$ contains terminal $t_i$. The Multiway Cut objective is to minimize the number or total weight of cut edges. For $k = 2$, the problem is equivalent to the minimum $st$-Cut problem. Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis proved that it is NP-complete and APX-hard for every $k > 2$ [9]. They also gave a simple combinatorial $(2 - 2/k)$-approximation algorithm. Later Călinescu, Karloff, and Rabani [7] showed how to obtain a $3/2$ approximation using linear programming. This


result was improved in a series of papers by Karger, Klein, Stein, Thorup, and Young [11], Buchbinder, Naor and Schwartz [5], and Sharma and Vondrák [13] (see also [6]). The currently best known approximation factor is 1.2965 [13]. The best known LP integrality gap and Unique Games Conjecture hardness is 1.20016 due to Bérczi, Chandrasekaran, Király, and Madan [4] (see also [2, 10, 12]).

In 2004, Svitkina and Tardos [14] introduced the Min-Max Multiway Cut problem. In this problem, as before, we need to partition graph $G$ into $k$ parts $P_1, \ldots, P_k$ so that each $P_i$ contains one terminal $t_i$. However, the objective function is different: Min-Max Multiway Cut asks to minimize the maximum of edge boundaries of sets $P_i$ i.e., minimize $\max_i \delta(P_i)$, where $\delta(P_i)$ is the total weight of edges crossing the cut $(P_i, V \setminus P_i)$. Svitkina and Tardos [14] gave an $O(\log^3 n)$ approximation algorithm for the problem. Later, Bansal, Feige, Krauthgamer, Makarychev, Nagarajan, Naor, and Schwartz [3] provided an $O(\sqrt{\log n \log k})$-approximation algorithm. Also, Ahmadi, Khuller, and Saha [1] studied a related Min-Max Multicut problem.

Recently, Chandrasekaran and Wang [8] proposed a common generalization of the Min Multiway Cut and Min-Max Multiway Cut problems, which they called Minimum $\ell_p$-norm Multiway Cut. This problem asks to minimize the $\ell_p$ norm of the edge boundaries of parts $P_1, \ldots, P_k$. In other words, the objective is to

$$\textbf{minimize: } \Big(\sum_{i=1}^{k} \delta(P_i)^p\Big)^{1/p}.$$

Note that this problem is equivalent to Min Multiway Cut when $p = 1$ and to Min-Max Multiway Cut when $p = \infty$. Chandrasekaran and Wang [8] gave an $O(\log^{3/2} n \log^{1/2} k)$ approximation for the problem. Further, they proved that the problem is NP-hard for every $p \geq 1$ and $k \geq 4$. Moreover, it does not admit an $O(k^{1-1/p-\varepsilon})$-approximation for every $\varepsilon > 0$ assuming the Small Set-Expansion Conjecture; a natural convex program for the problem has the intgerality gap of $\Omega(k^{1-1/p})$.

In this paper, we provide an improved $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation algorithm. We note that for $p = \infty$, our approximation guarantee matches the approximation of the algorithm due to Bansal et al. [3].[1] For $p = 2$, our approximation guarantee is $O(\log^{1/2} n \log k)$, which is $\Theta(\log n/\sqrt{\log k})$ times better than the approximation guarantee of the algorithm due to Chandrasekaran and Wang [8]. We also consider variants of Multiway Cut with norms other than the $\ell_p$ norm.

## 1.1 Our Results

We now formally state our results. First, we present an approximation algorithm for the $\ell_p$-norm Multiway Cut problem. We show that our algorithm achieves an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation for every $p > 1$.

▶ **Theorem 1.** *There exists a polynomial-time randomized algorithm that given a graph with $n$ vertices, $k$ terminals, and $p > 1$, finds an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation for $\ell_p$-norm Multiway Cut with high probability.*

Further, we provide approximation algorithms for Norm Multiway Cut with an arbitrary monotonic norm, a further generalization of $\ell_p$-norm Multiway Cut. The monotonic norm is defined as follows.

[1] Our algorithm is stated only for the case where $p$ is finite. However, we can solve an instance with $p = \infty$ by running the algorithm with $p = \log k$. Since $\|\cdot\|_{\log k}$ is within a constant factor of $\|\cdot\|_\infty$ for vectors in $\mathbb{R}^k$, this approach yields an $O(\sqrt{\log n} \log^{1/2+1/\log k} k) = O(\sqrt{\log n \log k})$-approximation.

▶ **Definition 2.** *A norm $\|\cdot\|$ on $\mathbb{R}^d$ is monotonic if for any $x, y \in \mathbb{R}^d$ with $|x_i| \le |y_i|$ for all $i \in [d]$, it holds $\|x\| \le \|y\|$.*

We consider two oracles to the monotonic norm used in the Norm Multiway Cut: (1) minimization oracle; (2) ordering oracle. For a set $A \subseteq [d]$, let $\mathbb{1}_A \in \{0,1\}^d$ denote the indicator vector of $A$, i.e., the $i$-th coordinate $(\mathbb{1}_A)_i = 1$ if $i \in A$; otherwise, $(\mathbb{1}_A)_i = 0$.

▶ **Definition 3.** *Given a monotonic norm $\|\cdot\|$ on $\mathbb{R}^d$, for any $i \in [d]$, the minimization oracle efficiently finds a set $A_i \subseteq [d]$ that minimizes the norm of indicator vectors among all subsets with size $i$, i.e.*

$$A_i = \operatorname*{arg\,min}_{A \subseteq [n], |A| = i} \|\mathbb{1}_A\|.$$

▶ **Definition 4.** *Given a monotonic norm $\|\cdot\|$ on $\mathbb{R}^d$, for any vector $x \in \mathbb{R}^d$, the ordering oracle efficiently finds an ordering of the vector $x$ that minimizes the norm, i.e.*

$$\pi_x = \operatorname*{arg\,min}_{\pi \in S_d} \|x^\pi\|,$$

*where $x^\pi$ denotes the ordering of $x$ regarding the permutation $\pi$.*

Assuming that they are given access to either a "minimization oracle" or a stronger "ordering oracle", our algorithms give $O(\log^{1/2} n \log^{7/2} k)$ and $O(\log^{1/2} n \log^{5/2} k)$ approximation, respectively. We remark that the oracles only answer queries about the norm and, in particular, there is an ordering oracle for the $\ell_p$-norm, weighted $\ell_p$-norm, and many other natural norms. Thus, our result implies an $O(\log^{1/2} n \log^{5/2} k)$ approximation for *weighted* $\ell_p$-norm Multiway Cut. We prove the following theorems in the full version of the paper.

▶ **Theorem 5.** *There exists a polynomial-time algorithm that for every monotonic norm with a minimization oracle, given a graph with $n$ vertices and $k$ terminals, finds an $O(\log^{1/2} n \log^{7/2} k)$ approximation for the Norm Multiway Cut with high probability.*

▶ **Theorem 6.** *There exists a polynomial-time algorithm that for every monotonic norm with an ordering oracle, given a graph with $n$ vertices and $k$ terminals, finds an $O(\log^{1/2} n \log^{5/2} k)$ approximation for the Norm Multiway Cut with high probability.*

Finally, we show that the problem becomes very hard if we are not given access to a norm minimization oracle. The proof is given in the full version of the paper.

▶ **Theorem 7.** *Consider the Norm Multiway Cut problem with a monotonic norm. Assume that the norm is given by a formula (in particular, we can easily compute the value of the norm; however, we are not given a minimization oracle for it). Then, assuming the Hypergraph Dense-vs-Random Conjecture, there is no polynomial-time algorithm for Norm Multiway Cut with approximation factor $\alpha(n) \le n^{1/4-\varepsilon}$ for every $\varepsilon > 0$.*

## 1.2 Proof Overview

We first describe our algorithm for the $\ell_p$-norm Multiway Cut. Our algorithm consists of three procedures: (1) covering procedure, (2) uncrossing procedure, and (3) aggregation procedure.

In the covering procedure, we generate a collection of subsets of the vertex set, $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$. We generate these sets iteratively by using a bi-criteria approximation algorithm for Unbalanced Terminal Cut by Bansal et al. [3] and a multiplicative weight

update method. See Section 2.1 and Algorithm 1 for details. Each set in $\mathcal{S}$ contains at most one terminal. These sets are not disjoint. While, these sets cover the entire graph, which means the union of all sets in $\mathcal{S}$ contains all vertices. The number of sets in $\mathcal{S}$ is at most $O(k \log n)$. We show that the $\ell_p$ norm and $\ell_1$ norm of the edge boundaries of sets in $\mathcal{S}$ is at most $O(\log^{1/p} n \cdot \alpha)\,\mathrm{OPT}$ and $O(\log n \cdot k^{1-1/p} \cdot \alpha)\,\mathrm{OPT}$ respectively, where $\alpha = \sqrt{\log n \log k}$ and OPT is the cost of the optimal solution. This covering procedure follows the approach by Bansal et al. [3] for Min-Max Multiway Cut. Chandrasekaran and Wang [8] also use a similar covering procedure for $\ell_p$-norm Multiway Cut. Their algorithm finds a cover $\mathcal{S}$ that satisfies the above properties except for the $\ell_1$ norm bound. We get this $\ell_1$ norm bound on the edge boundaries of sets in $\mathcal{S}$ by picking proper measure constraints for the Unbalanced Terminal Cut algorithm. This $\ell_1$ norm bound is important in the aggregation procedure to get an improved approximation.

Note that the sets in $\mathcal{S}$ are not disjoint. We use the uncrossing procedure to create a partition of the graph with at most $O(k \log k)$ sets. Our uncrossing procedure first sample $O(k \log k)$ sets from $\mathcal{S}$ uniformly at random. Then, we run an iterative uncrossing process given by Bansal et al. [3] over sampled sets until all sets are disjoint and have small boundaries. We show that all sampled sets cover almost the entire graph. The set of uncovered vertices does not contain terminals and has a small boundary with high probability. Next, we use the aggregation procedure to merge these $O(k \log k)$ sets into a $k$ partition. We assign $k$ sets containing one terminal to $k$ parts. For other sets without terminals, we assign them to $k$ parts almost uniformly such that each part has almost the same $\ell_1$ norm over assigned sets. After the uncrossing procedure, the $\ell_p$ norm and $\ell_1$ norm of edge boundaries is at most $O(\log^{1/p} k \cdot \alpha)\,\mathrm{OPT}$ and $O(k^{1-1/p} \cdot \alpha)\,\mathrm{OPT}$ respectively. We upper bound the sets containing one terminal and the sets with the largest edge boundary in each part by the above $\ell_p$ norm bound. For the remaining sets, by the $\ell_1$ norm bound and the uniform assignment, we upper bound the $\ell_p$ norm for these sets by $O(\alpha)\,\mathrm{OPT}$. Chandrasekaran and Wang [8] achieve an $O(\log n \cdot \alpha)$ where the $O(\log n)$ factor is due to their aggregation procedure. We use the $\ell_1$ norm bound in the covering procedure and a new aggregation procedure to reduce $O(\log n)$ extra factor to $O(\log^{1/p} n)$. We use the sampling in the uncrossing procedure to further reduce the extra factor from $O(\log^{1/p} n)$ to $O(\log^{1/p} k)$.

We now describe our algorithm for Norm Multiway Cut. We use the same framework with covering, uncrossing, and aggregation procedures. While, unlike the $\ell_p$ norm, the general monotonic norm may not be permutation invariant. For each terminal, we first compute a minimum cut that separates this terminal from other terminals. Then, we can remove all terminals and assign the remaining vertices freely among $k$ parts. We mainly use a bucketing idea to modify our algorithm. We partition $k$ coordinates into $\log_2 k$ buckets with exponentially increasing size $2^i$ such that the coordinates with large boundaries in the optimal solution are assigned to small buckets. Differing from the previous covering procedure, the new covering procedure uses the Unbalanced Terminal Cut algorithm with parameters related to each bucket. The cover $\mathcal{S}$ contains $O(2^i \log n \log k)$ sets in each bucket $i$. The boundary of every set in each bucket is relatively small, at most $O(\alpha)$ times the boundary of the optimal part in this bucket. We then run the uncrossing and aggregation procedure to create a multiway cut. We still sample each set in $\mathcal{S}$ with probability $\log_2 k / \log_2 n$. Thus, we have $O(2^i \log^2 k)$ sets in each bucket $i$ after the uncrossing procedure. For bucket $0 \le i \le \log_2 k$, we find a set of $2^i$ coordinates $I_i \in [k]$ that minimizes the norm of the indicator vector through the minimization oracle. We then assign $O(2^i \log^2 k)$ sets in each bucket to coordinates in $I_i$ such that each coordinate has $O(\log^2 k)$ sets in bucket $i$. Thus, we achieve an $O(\log^2 k \cdot \alpha)$ approximation for each bucket. Since these sets of coordinates $I_i$ may overlap, we lose an

additional $O(\log k)$ factor for $\log_2 k$ buckets. Suppose we have a stronger oracle that finds the best ordering for any given vector that minimizes the norm. Then, we provide an assignment for each bucket to avoid the large overlapping among buckets. Therefore, we avoid the extra $O(\log k)$ factor loss due to the overlapping.

## 2 $\ell_p$-norm Multiway Cut

In this section, we present our algorithm for $\ell_p$-norm Multiway Cut. We prove the following theorem. Our algorithm consists of three parts: covering procedure, uncrossing procedure, and aggregation procedure. We describe and analyze the covering procedure in Section 2.1, the uncrossing and aggregation procedures in Section 2.2.

▶ **Theorem 1.** *There exists a polynomial-time randomized algorithm that given a graph with $n$ vertices and $k$ terminals, and $p > 1$, finds an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation for the $\ell_p$-norm Multiway Cut with high probability.*

### 2.1 Covering Procedure

We first present and analyze a covering procedure in our algorithm. The covering procedure takes a undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$ and $k$ terminals $T = \{t_1, \dots, t_k\} \subset V$ as input and outputs a collection of sets $\mathcal{S} = \{S_1, \dots, S_m\}$ where $S_i \subset V$ for all $i$. All sets $S_i \in \mathcal{S}$ covers the entire graph, $\bigcup_{i=1}^m S_i = V$. Each set $S_i \in \mathcal{S}$ contains at most one terminal. For each subset $S \subseteq V$, we use $\partial(S) = E(S, V \setminus S)$ to denote all edges crossing the cut $(S, V \setminus S)$. We use $\delta(S) = \sum_{e \in \partial(S)} w(e)$ to denote the edge boundary of set $S$, which is the total weight of all edges crossing $(S, V \setminus S)$. We prove the following upper bounds on the $\ell_1$-norm and $\ell_p$-norm of the edge boundaries of these sets in $\mathcal{S}$, which is crucial for our approximation guarantee.

▶ **Lemma 8.** *Given a graph $G = (V, E)$ with $n$ vertices and $k$ terminals $T \subset V$, the covering procedure shown in Algorithm 1 returns $m = O(k \log n)$ sets $\mathcal{S} = \{S_1, \dots, S_m\}$ that satisfies*

1. $|S_i \cap T| \leq 1$ *for all* $i \in [m]$,
2. $\bigcup_{i=1}^m S_i = V$,
3. $\sum_{t=1}^m \delta(S_t)^p \leq \log n \cdot O(\alpha^p) \cdot \mathrm{OPT}^p$,
4. $\sum_{t=1}^m \delta(S_t) \leq \log n \cdot O(\alpha) \cdot k^{1-1/p} \cdot \mathrm{OPT}$,

*where $\alpha = \sqrt{\log n \log k}$ and* OPT *is the objective value of the optimal $\ell_p$-norm Multiway Cut.*

Our algorithm relies on an intermediate problem, Unbalanced Terminal Cut that we introduce now.

▶ **Definition 9** (Unbalanced Terminal Cut)**.** *The input to this problem is a tuple $\langle G, w, \mu, \rho, T \rangle$, where $G = (V, E)$ is a graph with edge weights $w : E \to \mathbb{R}_{\geq 0}$, a measure $\mu : V \to \mathbb{R}_{\geq 0}$, a parameter $\rho \in (0, 1]$, and a set of terminals $T$. The goal is to find $S \subseteq V$ of minimum cost $\delta(S)$ satisfying:*

1. $|S \cap T| \leq 1$,
2. $\mu(S) \geq \rho \cdot \mu(V)$.

Bansal, Feige, Krauthgamer, Makarychev, Nagarajan, Naor, and Schwartz [3] gave a bi-criteria approximation algorithm for Unbalanced Terminal Cut that we state in the following theorem.

**Algorithm 1** Covering Procedure.

Set $t = 1$, and $\mu_1(v) = 1$ for all $v \in V$. Let $\mathcal{S} = \varnothing$.
**while** $\sum_{v \in V} \mu_t(v) \geq \frac{1}{n}$ **do**
  Let $P_i^*$ be a set as stated in Lemma 14.
  Guess $\mu_t(P_i^*)$.
  Let $S_t \subseteq V$ be the solution for Unbalanced Terminal Cut instance $\langle G, w, \mu_t, \max\{\frac{1}{2k}, \frac{\mu_t(P_i^*)}{\mu_t(V)}\}, T\rangle$.
  Let $\mathcal{S} = \mathcal{S} \cup \{S_t\}$.
  **for** $v \in V$ **do**
    Set $\mu_{t+1}(v) = \begin{cases} \mu_t(v)/2, & \text{if } v \in S_t, \\ \mu_t(v), & \text{if } v \notin S_t. \end{cases}$
  Set $t = t + 1$
**return** $\mathcal{S}$.

▶ **Theorem 10.** *There exists a polynomial-time algorithm that given an instance $\langle G, w, \mu, \rho, T\rangle$ of Unbalanced Terminal Cut, finds a set $S \subseteq V$ satisfying $|S \cap T| \leq 1$, $\mu(S) \geq \Omega(\rho) \cdot \mu(V)$, and $\delta(S) \leq \alpha \cdot \mathrm{OPT}_{\langle G,w,\mu,\rho,T\rangle}$ where $\alpha = O\left(\sqrt{\log n \log 1/\rho}\right)$.*

Our covering procedure relies on the multiplicative weights update method and is inspired by the algorithm in [3]. It initializes the measure of each vertex to one. At each iteration $t$, the algorithm guesses the measure $\mu_t(P_i^*)$ of a particular set $P_i^*$ in an optimal solution and computes $S_t$ of measure $\mu_t(S_t) \approx \mu_t(P_i^*)$ using the Unbalanced Terminal Cut algorithm in Theorem 10. The existence of such a $P_i^*$ is shown in Lemma 14. Once $S_t$ is computed, the algorithm decreases the measure of the vertices covered by $S_t$ by a factor of 2. The algorithm terminates when the total measure of vertices is less than $1/n$.

We guess $\mu_t(P_i^*)$ as follows: For any set $S \subseteq V$, its measure $\mu_t(S)$ lies in the range $[\mu_t(u), n \cdot \mu_t(u)]$, where $u = \arg\max_{v \in S} \mu_t(v)$ is the heaviest vertex in $S$. Thus $\mu_t(P_i^*)$ can be well approximated by the set $A = \left\{2^i \cdot \mu_t(v) : v \in V, i = 0, \ldots, \lfloor \log_2 n \rfloor\right\}$ of size $O(n \log n)$. For each candidate $a \in A$ we compute a set $S(a)$ using the Unbalanced Terminal Cut algorithm with a parameter $a$ and choose $S_t = \arg\min_{a \in A} \delta(S(a))$ with the smallest cost. We give a pseudo-code for this algorithm in Algorithm 1. We remark that one can think of this algorithm as of multiplicative weight update algorithm for solving a covering LP with constraints from Lemma 8.

We then analyze this covering algorithm in Algorithm 1. Let $\mathcal{S} = \{S_1, \cdots, S_m\}$ denote the collection of $m$ sets output by Algorithm 1. By Theorem 10, every set $S_i$ contains at most one terminal. First, for any fixed vertex $v \in V$, we give a lower bound on the number of sets containing $v$.

▷ Claim 11. For a vertex $v \in V$, let $N_v = |\{S_t |\ v \in S_t\}|$ denote the number of sets containing $v$. Then $N_v \geq \Omega(\log n)$.

Proof. Recall that initially $\mu_1(v) = 1$ and after iteration $m$ its measure becomes $\mu_{m+1}(v) = \left(\frac{1}{2}\right)^{N_v}$. Due to the stopping condition of our algorithm we have $\mu_{m+1}(V) < \frac{1}{n}$. Thus, $\frac{1}{2^{N_v}} < \frac{1}{n}$ and the claim follows. ◁

Next, we bound the number of sets in $\mathcal{S}$. In the following claim we give an upper bound on the total normalized measure of the sets produced by our algorithm.

▷ Claim 12. $\sum_{t=1}^{m} \frac{\mu_t(S_t)}{\mu_t(V)} \leq 4 \ln n + 1$.

Proof. Observe that the total measure at iteration $t$ can be described as follows:

$$\mu_t(V) = \mu_{t-1}(V) - \frac{\mu_{t-1}(S_{t-1})}{2} = \mu_{t-1}(V) \cdot \left(1 - \frac{\mu_{t-1}(S_{t-1})}{2\mu_{t-1}(V)}\right).$$

Since $\mu_m(V) \geq \frac{1}{n}$, we have

$$\frac{1}{n} \leq \mu_m(V) = \mu_1(V) \cdot \prod_{t=1}^{m-1} \left(1 - \frac{\mu_t(S_t)}{2\mu_t(V)}\right) \leq n \cdot \prod_{t=1}^{m-1} e^{-\frac{\mu_t(S_t)}{2\mu_t(V)}} = n \cdot e^{-\frac{1}{2}\cdot\sum_{t=1}^{m-1}\frac{\mu_t(S_t)}{\mu_t(V)}},$$

which implies $\sum_{t=1}^{m-1} \frac{\mu_t(S_t)}{\mu_t(V)} \leq 4\ln n$. Since $\frac{\mu_m(S_m)}{\mu_m(V)} \leq 1$, we get the desired result. ◁

We obtain an upper-bound on the number of sets in $\mathcal{S}$ which immediately follows from Claim 12 and the fact that $\mu_t(S_t) \geq \Omega(1/k)\mu_t(V)$ for all $t$.

▶ **Corollary 13.** *The cover $\mathcal{S}$ returned by Algorithm 1 contains $m = O(k \log n)$ sets.*

We prove the existence of a set in an optimal solution with large measure and small cut value.

▶ **Lemma 14.** *Let $\mathcal{P}^* = (P_1^*, \dots, P_k^*)$ be an optimal solution to an $\ell_p$-norm Multiway Cut instance and let* OPT *denote the $\ell_p$-norm of $\mathcal{P}^*$. For any measure $\mu : V \to \mathbb{R}_{\geq 0}$ on vertices such that $\mu(V) \neq 0$, there exists an $i \in [k]$ such that the following three conditions hold:*

1. $\delta(P_i^*)^p \leq 5 \cdot \mathrm{OPT}^p \cdot \frac{\mu(P_i^*)}{\mu(V)}$
2. $\delta(P_i^*) \leq 5k^{1-1/p} \cdot \mathrm{OPT} \cdot \frac{\mu(P_i^*)}{\mu(V)}$
3. $\mu(P_i^*) \geq \frac{\mu(V)}{2k}$

**Proof.** Let

$$J = \{j \in [k] : \delta(P_j^*)^p \leq 5 \cdot \mathrm{OPT}^p \cdot {}^{\mu(P_j^*)}\!/_{\mu(V)}, \delta(P_j^*) \leq 5k^{1-1/p} \cdot \mathrm{OPT} \cdot {}^{\mu(P_j^*)}\!/_{\mu(V)}\}$$

be the indices of sets in $\mathcal{P}^*$ that satisfies conditions 1 and 2 in Lemma 14. It is sufficient to show that $\sum_{j\in[k]\setminus J} \mu(P_j^*) < \mu(V)/2$. If $\sum_{j\in[k]\setminus J} \mu(P_j^*) < \mu(V)/2$, then there exists a $j \in J$ such that $\mu(P_j^*) \geq {}^{\mu(V)}\!/_{2k}$, which implies this set $P_j^*$ satisfies all three conditions.

We now show that $\sum_{j\in[k]\setminus J} \mu(P_j^*) < \mu(V)/2$. Let $\bar{J}_1 = \{j \in [k] \setminus J : \delta(P_j^*)^p > 5 \cdot \mathrm{OPT}^p \cdot {}^{\mu(P_j^*)}\!/_{\mu(V)}\}$ be the indices of sets $P_j^*$ that does not satisfy condition 1. Let $\bar{J}_2 = \{j \in [k] \setminus J : \delta(P_j^*) > 5k^{1-1/p} \cdot \mathrm{OPT} \cdot {}^{\mu(P_j^*)}\!/_{\mu(V)}\}$ be the indices of sets $P_j^*$ that does not satisfy condition 2. Note that $[k] \setminus J = \bar{J}_1 \cup \bar{J}_2$. Then, we have

$$\begin{aligned}
\sum_{j\in[k]\setminus J} \mu(P_j^*) &\leq \sum_{j\in\bar{J}_1} \mu(P_j^*) + \sum_{j\in\bar{J}_2} \mu(P_j^*) \\
&\leq \sum_{j\in\bar{J}_1} \mu(V) \cdot \frac{\delta(P_j^*)^p}{5\,\mathrm{OPT}^p} + \sum_{j\in\bar{J}_2} \mu(V) \cdot \frac{\delta(P_j^*)}{5k^{1-1/p}\,\mathrm{OPT}} \\
&\leq \mu(V) \cdot \sum_{j\in[k]} \left(\frac{\delta(P_j^*)^p}{5\,\mathrm{OPT}^p} + \frac{\delta(P_j^*)}{5k^{1-1/p}\,\mathrm{OPT}}\right).
\end{aligned}$$

Since $\mathcal{P}^*$ is a partition with an optimal cost, we have

$$\sum_{i=1}^{k} \delta(P_i^*)^p = \mathrm{OPT}^p.$$

Similarly, we have

$$k^{-1/p} \cdot \mathrm{OPT} = \left( \sum_{i=1}^{k} \frac{1}{k} \cdot \delta(P_i^*)^p \right)^{1/p} \geq \frac{1}{k} \sum_{i=1}^{k} \delta(P_i^*),$$

where the inequality follows from Jensen's inequality. Thus, we have

$$\sum_{j \in [k] \setminus J} \mu(P_j^*) \leq \mu(V) \cdot \sum_{j \in [k]} \frac{\delta(P_j^*)^p}{5\,\mathrm{OPT}^p} + \frac{\delta(P_j^*)}{5k^{1-1/p}\,\mathrm{OPT}} \leq \mu(V) \cdot \frac{2}{5} < \frac{\mu(V)}{2}. \qquad \blacktriangleleft$$

We now prove the main lemma in this section. Specifically, we give two upper-bounds on the $\ell_1$-norm and $\ell_p$-norm of the cut values of the sets produced by the covering procedure in Algorithm 1, respectively.

**Proof of Lemma 8.** We already show the number of sets in $\mathcal{S}$ is at most $m = O(k \log n)$ in Corollary 13. By Theorem 10 and Claim 11, we have every $S_i$ contains at most one terminal and all sets in $\mathcal{S}$ covers the entire graph. Thus, it is sufficient to prove the two bounds on the $\ell_1$-norm and $\ell_p$-norm of the cut values of the sets in $\mathcal{S}$ as shown in Conditions 3 and 4 in the lemma.

Due to Lemma 14 at each iteration $t$, there exists a set $P_i^*$ in an optimal solution with a measure $\mu_t(P_i^*) \geq \frac{\mu_t(V)}{2k}$ such that

$$\delta(P_i^*) \leq 5 \cdot \min \left\{ \left( \frac{\mu_t(P_i^*)}{\mu_t(V)} \right)^{1/p}, k^{1-1/p} \cdot \frac{\mu_t(P_i^*)}{\mu_t(V)} \right\} \cdot \mathrm{OPT}\,.$$

Thus, at each iteration $t$ of Algorithm 1, we have

$$\delta(S_t) \leq O(\alpha) \cdot \min \left\{ \left( \frac{\mu_t(P_i^*)}{\mu_t(V)} \right)^{1/p}, k^{1-1/p} \cdot \frac{\mu_t(P_i^*)}{\mu_t(V)} \right\} \cdot \mathrm{OPT}\,.$$

Note that each set $S_t$ is computed by the Unbalanced Terminal Cut algorithm in Theorem 10. Thus, we have $\mu_t(S_t) \geq \Omega(\mu_t(P_i^*))$. Since $\mu_t(S_t) \geq \Omega(\mu_t(P_i^*))$ holds, we obtain (1) $\delta(S_t)^p \leq O(\alpha^p) \cdot \mathrm{OPT}^p \cdot \frac{\mu_t(S_t)}{\mu_t(V)}$; and (2) $\delta(S_t) \leq O(\alpha) \cdot k^{1-1/p} \cdot \mathrm{OPT} \cdot \frac{\mu_t(S_t)}{\mu_t(V)}$. These provide

$$\sum_{t=1}^{m} \delta(S_t)^p \leq O(\alpha^p) \cdot \mathrm{OPT}^p \cdot \sum_{t=1}^{m} \frac{\mu_t(S_t)}{\mu_t(V)},$$
$$\sum_{t=1}^{m} \delta(S_t) \leq O(\alpha) \cdot k^{1-1/p} \cdot \mathrm{OPT} \cdot \sum_{t=1}^{m} \frac{\mu_t(S_t)}{\mu_t(V)}.$$

By Claim 12, we have $\sum_{t=1}^{m} \frac{\mu_t(S_t)}{\mu_t(V)} = O(\log n)$. Then, we get the desired upper bounds on $\sum_{t=1}^{m} \delta(S_t)^p$ and $\sum_{t=1}^{m} \delta(S_t)$. $\blacktriangleleft$

## 2.2 Uncrossing and Aggregation Procedures

In this section, we provide procedures that transform the cover of the graph $\mathcal{S}$ produced by the covering procedure into a partition of the graph $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$. Each set $P_i$ in $\mathcal{P}$ contains exactly one terminal in $T$. With a positive probability, this solution is an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation for the $\ell_p$-Norm Multiway Cut.

▶ **Theorem 15.** *Given a graph $G = (V, E)$ and $k$ terminals $T \subset V$, there exists a polynomial-time algorithm that returns a partition of the graph $\mathcal{P} = \{P_1, P_2 \ldots, P_k\}$ such that with probability at least $3/4 - 1/k$*

1. $|P_i \cap T| = 1$ *for all* $i \in [k]$,
2. $\left(\sum_{i=1}^{k} \delta(P_i)^p\right)^{1/p} \leq O(\log^{1/2} n \log^{1/2+1/p} k) \cdot \mathrm{OPT}$.

Note that the sets in the cover $\mathcal{S}$ are not disjoint. We first use the uncrossing procedure to generate a $m' = O(k \log k)$ partition of the graph $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_{m'}\}$ from the cover $\mathcal{S}$ produced by the covering procedure. We sample $O(k \log k)$ sets from $\mathcal{S}$ uniformly at random. These sampled sets cover a large fraction of the graph. Then, we generate disjoint sets from these sampled sets by using the uncrossing step in [3]. The uncrossing procedure is shown in Algorithm 2. We then merge these sets in $\mathcal{P}'$ to get a $k$-partition $\mathcal{P}$ using the aggregation procedure in Algorithm 3.

In the aggregation procedure, we assign all sets in $\mathcal{P}'$ into $k$ parts to get a $k$-partition. Since $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_{m'}\}$ is a partition of the graph and each set $P'_i$ contains at most one terminal, there are exactly $k$ sets containing one terminal in $\mathcal{P}'$. Suppose $P'_1, P'_2, \cdots, P'_k$ are these sets containing one terminal. We initially assign these sets $P'_1, P'_2, \cdots, P'_k$ to $k$ parts $P_1, P_2, \ldots, P_k$. Let $\mathcal{Q} = \mathcal{P}' \setminus \{P_1, P_2, \cdots, P_k\}$ be the sets in $\mathcal{P}'$ that does not contain any terminals. We assign all sets in $\mathcal{Q}$ into $k$ parts in a round-robin approach. We sort the sets in $\mathcal{Q}$ by the cut values in descending order and denote it by $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_{m'-k}\}$. We then partition all sets in $\mathcal{Q}$ into $k$ buckets $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ as follows. Consider every $k$ consecutive sets $\{Q_{jk+1}, Q_{jk+2}, \cdots, Q_{(j+1)k}\}$ in $\mathcal{Q}$ for $0 \leq j \leq \lfloor m'-k/k \rfloor$. If $jk + i > n$ for $j = \lfloor m'-k/k \rfloor$ and some $i \in [k]$, then let $Q_{jk+i} = \varnothing$. For every $i \in [k]$, we assign the set $Q_{jk+i}$ to the bucket $\mathcal{Q}_i$. Finally, we assign each bucket $\mathcal{Q}_i$ to part $P_i$ and set $P_i = P_i \cup (\bigcup_{Q_j \in \mathcal{Q}_i} Q_j)$.

**Algorithm 2** Uncrossing Procedure.

Sample $m'' - 1 = 12k \ln k$ sets $\mathcal{S}' = (S'_1, S'_2, \ldots, S'_{m''-1})$ from $\mathcal{S}$ uniformly at random.
Sort sets in $\mathcal{S}'$ in a random order.
Set $P'_i = S'_i \setminus \cup_{j<i} S'_j$ for all $i = 1, 2 \ldots, m'' - 1$.
**while** there exists a set $P'_i$ such that $\delta(P'_i) > 2\delta(S'_i)$ **do**
  Set $P'_i = S'_i$ and for all $j \neq i$, $P'_j = P'_j \setminus S'_i$.
Set the set $P'_{m''} = V \setminus \cup_{i=1}^{m''-1} P'_i$.
**return** all non-empty sets $P'_i$.

**Algorithm 3** Aggregation Procedure.

Set $\mathcal{P} = \{P'_i \in \mathcal{P}' : P'_i \cap T \neq \varnothing\} = \{P_1, P_2, \ldots, P_k\}$.
Set $\mathcal{Q} = \mathcal{P}' \setminus \mathcal{P}$.
Sort the sets in $\mathcal{Q} = \{Q_1, \ldots, Q_{m'-k}\}$ by the cut value in descending order.
Partition the sets in $\mathcal{Q}$ into $k$ buckets $\mathcal{Q}_1, \cdots, \mathcal{Q}_k$, where

$$\mathcal{Q}_i = \{Q_j \in \mathcal{Q} : (j-1) \mod k = i - 1\}.$$

Set $P_i = (\bigcup_{Q_j \in \mathcal{Q}_i} Q_j) \cup P_i$ for all $i = 1, \ldots, k$.
**return** all sets $P_1, P_2, \ldots, P_k$.

We first prove the following lemma on the partition $\mathcal{P}'$ returned by the uncrossing procedure.

▶ **Lemma 16.** *Let $\mathcal{S}$ denote the collection of sets produced by the covering procedure for a graph $G = (V, E)$ and $k$ terminals $T$. Given $\mathcal{S}$ as input, the uncrossing procedure as shown in Algorithm 2 generates a $m' = O(k \log k)$ partition of the graph $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_{m'}\}$ such that with probability at least $3/4 - 1/k$*

1. $|P'_i \cap T| \leq 1$ *for all* $i \in [m']$,
2. $\sum_{i=1}^{m'} \delta(P'_i)^p \leq O(\log k \cdot \alpha^p) \cdot \mathrm{OPT}^p$,
3. $\sum_{i=1}^{m'} \delta(P'_i) \leq O(k^{1-1/p} \cdot \alpha) \cdot \mathrm{OPT}$,

*where $\alpha = \sqrt{\log n \log k}$.*

**Proof.** We consider all sets $P'_1, P'_2, \ldots, P'_{m''}$ generated in the uncrossing procedure (Algorithm 2), including those empty sets that are not returned. If the set $P'_i$ is empty, we take $\delta(P'_i) = 0$. It is easy to see that these sets $P'_1, P'_2, \ldots, P'_{m''}$ are disjoint, and $\bigcup_{i=1}^{m''} P'_i = V$.

We first show that Algorithm 2 terminates in polynomial time. In graph $G$, we assume that the ratio between the largest non-infinite edge weight $w_{max}$ and the smallest non-zero edge weight $w_{min}$ is at most $w_{max}/w_{min} \leq n^2/\varepsilon$ for a small constant $\varepsilon > 0$. If the graph does not satisfy this assumption, then we transform it into an instance satisfying this condition as follows. We guess the largest weight of the cut edge in the optimal solution, denoted by $W$. There are at most $O(n^2)$ different edge weights. Then, we construct a new graph $G'$ with the same vertex set $V$ and edge set $E$. For every edge $e \in E$, we assign its weight $w'(e)$ in $G'$ to be $w(e)$ if $\varepsilon W/n^2 \leq w(e) \leq W$, $w'(e) = 0$ if $w(e) < \varepsilon W/n^2$, and $w'(e) = \infty$ if $w(e) \geq W$. Thus, the new graph $G'$ satisfies the assumption that $w_{max}/w_{min} \leq n^2/\varepsilon$. Let $\mathrm{OPT}'$ be the optimal value of $\ell_p$ multiway cut on graph $G'$. We know that $\mathrm{OPT}' \leq \mathrm{OPT}$ since the optimal multiway cut on graph $G$ has a smaller value on graph $G'$. Suppose we find an $\alpha$-approximation for $\ell_p$ multiway cut on graph $G'$. Then, the same partition on the original graph $G$ has an objective value at most $\alpha \cdot \mathrm{OPT}' + \varepsilon W \leq (\alpha + \varepsilon)\,\mathrm{OPT}$. Hence, this $\alpha$-approximation solution on $G'$ provides an $(\alpha + \varepsilon)$-approximation on $G$.

Consider any iteration of Algorithm 2. Let $P'_i$ be the partition of $V$ before the current uncrossing iteration. Suppose we pick a set $P'_i$ such that $\delta(P'_i) > 2\delta(S'_i)$. For any two subsets $A, B \subseteq V$, we use $\delta(A, B)$ to denote the total weight of edges crossing $A$ and $B$. Then, we have the $\ell_1$-norm of the cut values after this iteration is

$$\begin{aligned}
\delta(S'_i) + \sum_{j \neq i} \delta(P'_j \setminus S'_i) &\leq \delta(S'_i) + \sum_{j \neq i} \delta(P'_j) - \delta(P'_j, S'_i \setminus P'_j) + \delta(S'_i, P'_j \setminus S'_i) \\
&\leq \delta(S'_i) - \delta(P'_i) + \delta(S'_i) + \sum_{j \neq i} \delta(P'_j) \\
&\leq 2\delta(S'_i) - 2\delta(P'_i) + \sum_j \delta(P'_j) \leq \sum_j \delta(P'_j) - 2w_{min},
\end{aligned}$$

where the last inequality is due to $\delta(P'_i) > 2\delta(S'_i)$ and the minimum non-zero edge weight is $w_{min}$. Thus, the $\ell_1$-norm of the cut values decreases by $2w_{min}$ after each iteration. Since the largest $\ell_1$-norm of the cut values is at most $w_{max}n^2$, the total number of iterations is polynomial in $n$.

We then show that the partition returned by Algorithm 2 satisfies two conditions in the Lemma. We first show that each set $P'_i$ contains at most one terminal. Note that for every $i = 1, 2, \ldots, m'' - 1$, the set $P'_i$ is a subset of $S'_i \in \mathcal{S}$. By Lemma 8, we have

$|P'_i \cap T| \le |S'_i \cap T| \le 1$ for all $i = 1, 2, \ldots, m'' - 1$. By Claim 11, every vertex $u \in V$ is covered by at least $\log_2 n$ sets in $\mathcal{S}$. By Corollary 13, the cover $\mathcal{S}$ contains at most $6k \log_2 n$ sets. Thus, a random set in $\mathcal{S}$ covers $u$ with probability at least $1/6k$. For each vertex $u \in V$, the probability that $u$ is not covered by any set in $\mathcal{S}'$ is at most

$$\mathbb{P}\{u \notin \cup_{i=1}^{m''-1} S'_i\} \le \left(1 - \frac{1}{6k}\right)^{12k \ln k} \le \frac{1}{k^2}. \tag{1}$$

By the union bound over all terminals, all terminals are covered by $\mathcal{S}'$ with probability at least $1 - 1/k$. Thus, the set $P'_{m''}$ contains no terminal with probability at least $1 - 1/k$.

We now bound the $\ell_1$-norm of the cut values of sets $P'_1, P'_2, \ldots, P'_{m''}$. The first two steps in Algorithm 2 can be implemented equivalently by sorting sets in $\mathcal{S}$ in a random order and picking the first $m'' - 1 = 12k \ln k$ sets as $\mathcal{S}'$. Let $S_1, S_2, \ldots, S_m$ be the sets in $\mathcal{S}$ in a random order. Let $\tilde{P}_i = S'_i \setminus \cup_{j<i} S'_j$ for $i = 1, 2, \ldots, m$. Then, for any $i = 1, 2, \ldots, m'' - 1$, the set $\tilde{P}_i$ corresponds to the set $P'_i$ before running the while loop in Algorithm 2.

We first bound the expected $\ell_1$-norm of the cut values of sets $\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_{m''-1}$. Note that we have

$$\sum_{i=1}^{m''-1} \delta(\tilde{P}_i) \le \sum_{i=1}^{m} \delta(\tilde{P}_i).$$

We assign each cut edges $(u, v) \in \partial \tilde{P}_i$ into the following two types: (1) edge $(u, v)$ is cut by a set $\tilde{P}_j$ for $j < i$; (2) edge $(u, v)$ is first cut by the set $\tilde{P}_i$. Let $E_i$ be the set of cut edges that first cut by the set $\tilde{P}_i$. Let $w(E_i) = \sum_{e \in E_i} w(e)$ be the total weight of edges in $E_i$. Each cut edge is counted twice in $\sum_{i=1}^{m} \delta(\tilde{P}_i)$, while each cut edge is counted exactly once in $\sum_{i=1}^{m} w(E_i)$. Thus, we have

$$\sum_{i=1}^{m} \delta(\tilde{P}_i) = 2 \sum_{i=1}^{m} w(E_i).$$

Note that $E_i \subseteq \partial S_i$ is a subset of edges cut by $S_i$. Each edge $(u, v) \in \partial S_i$ is a cut edge in $E_i$ after uncrossing if and only if $S_i$ is the first set among all sets that contain node $u$ or node $v$ in the uncrossing sequence. Suppose $S_i$ only contains node $u$. Then, the probability that $(u, v)$ is contained in $E_i$ is at most the probability that $S_i$ is the first set among all the sets that contain node $u$ in the uncrossing sequence. If a set in $\mathcal{S}$ that contains node $v$ is before set $S_i$, then this edge $(u, v)$ is not count in $E_i$. By Claim 11, we have

$$\mathbb{P}\{(u, v) \in E_i\} \le \mathbb{P}\{S_i \text{ is the first set that contains } u\} \le \frac{1}{\log_2 n}.$$

Therefore, we have the expected $\ell_1$-norm of the cut values of sets $\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_m$ is at most

$$\begin{aligned}\mathbf{E}\left[\sum_{i=1}^{m} \delta(\tilde{P}_i)\right] &= 2 \sum_{i=1}^{m} \mathbf{E}[w(E_i)] = 2 \sum_{i=1}^{m} \sum_{e \in \partial S_i} w(e) \cdot \mathbb{P}\{e \in E_i\} \\ &\le \frac{2}{\log_2 n} \sum_{i=1}^{m} \delta(S_i) \le k^{1-1/p} \cdot O(\alpha) \cdot \mathrm{OPT},\end{aligned}$$

where the last inequality is from Lemma 8. At every iteration of the while loop, the $\ell_1$-norm of the cut values of sets $P'_1, P'_2, \cdots, P'_{m''-1}$ only decreases. Thus, we have

$$\mathbf{E}\left[\sum_{i=1}^{m''-1} \delta(P'_i)\right] \le \mathbf{E}\left[\sum_{i=1}^{m''-1} \delta(\tilde{P}_i)\right] \le k^{1-1/p} \cdot O(\alpha) \cdot \mathrm{OPT}\,.$$

Thus, the expected $\ell_1$-norm of the cut values of sets $P'_1, P'_2, \ldots, P'_{m''}$ is

$$\mathbf{E}\left[\sum_{i=1}^{m''} \delta(P'_i)\right] \leq 2 \cdot \mathbf{E}\left[\sum_{i=1}^{m''-1} \delta(P'_i)\right] \leq k^{1-1/p} \cdot O(\alpha) \cdot \mathrm{OPT}\,.$$

To bound the $\ell_p$-norm of edge boundaries, we then bound the edge boundary of the last set $P'_{m''}$. We only consider the subsampling process in the uncrossing procedure. We sample $O(k \log k)$ sets from the cover $\mathcal{S}$ uniformly at random. Consider every edge $(u, v)$ in the boundary of sets in cover $\mathcal{S}$. If this edge $(u, v)$ is a cut edge crossing $P'_{m''}$ and $v \in P'_{m''}$, then one of the sets $S_i \in \mathcal{S}$ that contains node $u$ is sampled and node $v$ is not covered by sampled sets. Each set $S_i \in \mathcal{S}$ is sampled with probability $O(\log k / \log n)$. Suppose the set $S_i \in \mathcal{S}$ cuts this edge $(u, v)$ and contains node $u$. Similar to Equation (1), the probability that node $v \in P'_{m''}$ conditioned on $S_i \in \mathcal{S}'$ is at most $2/k^2$. Thus, we have

$$\begin{aligned}
\mathbf{E}[\delta(P'_{m''})] &= \mathbf{E}\left[w\left\{(u,v) \in \bigcup_{i=1}^{m''-1} \partial S'_i : u \notin P'_{m''} \text{ and } v \in P'_{m''}\right\}\right] \\
&\leq \sum_{i=1}^{m} \sum_{(u,v)\in\partial S_i} w(u,v) \cdot \mathbb{P}\{u \in S_i, S_i \in \mathcal{S}', v \in P'_{m''}\} \\
&\leq O\left(\frac{1}{k^2} \cdot \frac{\log k}{\log n}\right) \cdot \sum_{i=1}^{m} \delta(S_i) \leq O(\alpha) \cdot \mathrm{OPT},
\end{aligned}$$

where the last inequality is due to condition 4 in Lemma 8.

After the while loop, we have $\delta(P'_i) \leq 2\delta(S'_i)$ for all $i = 1, 2, \ldots, m''-1$. Since $\mathbf{E}[\delta(P'_{m''})] \leq O(\alpha) \cdot \mathrm{OPT}$, by Markov's Inequality, we have with probability at least $7/8$ that $\delta(P'_{m''}) \leq O(\alpha) \cdot \mathrm{OPT}$. Since we subsample a fraction $O(\log k / \log n)$ of sets in the cover $S$ uniformly at random, we have

$$\mathbf{E}\left[\sum_{i=1}^{m''-1} \delta(S'_i)^p\right] \leq O\left(\frac{\log k}{\log n}\right) \sum_{i=1}^{m} \delta(S_i)^p.$$

When $\delta(P'_{m''}) \leq O(\alpha) \cdot \mathrm{OPT}$, we have

$$\begin{aligned}
\mathbf{E}\left[\sum_{i=1}^{m''} \delta(P'_i)^p\right] &\leq 2^p \cdot \mathbf{E}\left[\sum_{i=1}^{m''-1} \delta(S'_i)^p\right] + \mathbf{E}\delta(P'_{m''})^p \\
&\leq 2^p \cdot O\left(\frac{\log k}{\log n}\right) \sum_{i=1}^{m} \delta(S_i)^p + O(\alpha^p) \cdot \mathrm{OPT}^p \leq O(\log k \cdot \alpha^p) \cdot \mathrm{OPT}^p,
\end{aligned}$$

where the third inequality is from the condition 3 in Lemma 8. Therefore, we have the conditions 2 and 3 in this lemma hold in expectation with probability at least $7/8$. By Markov's Inequality, we have the conditions 2 and 3 in the lemma hold simultaneously with probability at least $3/4$. Since the condition 1 hold with probability at least $1 - 1/k$, we have all conditions hold with probability at least $3/4 - 1/k$. ◀

Next, we analyze the aggregation procedure, which merges these sets to get a $k$ partition of the graph.

**Proof of Theorem 15.** By Lemma 16, the partition $P'_1, P'_2, \ldots, P'_{m'}$ returned by the uncrossing procedure (Algorithm 2) satisfies the following three conditions with probability at least $3/4 - 1/k$:

1. $|P'_i \cap T| \le 1$ for all $i \in [m']$,
2. $\sum_{i=1}^{m'} \delta(P'_i)^p \le O(\log k \cdot \alpha^p) \cdot \mathrm{OPT}^p$,
3. $\sum_{i=1}^{m'} \delta(P'_i) \le O(k^{1-1/p} \cdot \alpha) \cdot \mathrm{OPT}$.

We now assume the partition $\mathcal{P}' = \{P'_1, P'_2, \dots, P'_{m'}\}$ given by the uncrossing procedure satisfies these three conditions. Then, we use the aggregation procedure as shown in Algorithm 3 on this partition $\mathcal{P}' = \{P'_1, P'_2, \dots, P'_{m'}\}$ to get a $k$-partition $\mathcal{P} = \{P_1, P_2, \cdots, P_k\}$. Since each part $P_i$ has exactly one set $P'_i$ containing one terminal, we have $|P_i \cup T| = 1$ for all $i \in [k]$.

We now bound the $\ell_p$-norm of the cut values. Let $Q'_i = \bigcup_{j>k, Q_j \in \mathcal{Q}_i} Q_j$ be the union of sets in bucket $\mathcal{Q}_i$ excluding the set with the largest cut in that bucket. Thus, we have each part $P_i = Q'_i \cup Q_i \cup P'_i$ for all $i \in [k]$. By the triangle inequality, we have

$$\left(\sum_{i=1}^{k} \delta(P_i)^p\right)^{1/p} \le \left(\sum_{i=1}^{k} \delta(Q'_i)^p\right)^{1/p} + \left(\sum_{i=1}^{k} \delta(Q_i)^p\right)^{1/p} + \left(\sum_{i=1}^{k} \delta(P'_i)^p\right)^{1/p}.$$

By Lemma 16, the $\ell_p$-norm of the cut values of sets $P'_1, P'_2, \dots, P'_k$ is

$$\left(\sum_{i=1}^{k} \delta(P'_i)^p\right)^{1/p} \le O(\log^{1/p} k \cdot \alpha) \cdot \mathrm{OPT}\,.$$

Similarly, we have the $\ell_p$-norm of the cut values of sets $Q_1, Q_2, \dots, Q_k$ is

$$\left(\sum_{i=1}^{k} \delta(Q_i)^p\right)^{1/p} \le O(\log^{1/p} k \cdot \alpha) \cdot \mathrm{OPT}\,.$$

We then bound the $\ell_p$-norm of the cut values of sets $Q'_1, Q'_2, \dots, Q'_k$. We first bound the cut value of each set $Q'_i$. Since $Q_i$ are sorted by the cut value in descending order, we have

$$\delta(Q'_i) \le \sum_{j>k, Q_j \in \mathcal{Q}_i} \delta(Q_j) \le \sum_{Q_j \in \mathcal{Q}_k} \delta(Q_j) \le \frac{1}{k} \sum_{Q_j \in \mathcal{Q}} \delta(Q_j),$$

where the second inequality is due to $\delta(Q_{i+zk}) \le \delta(Q_{zk})$ for $z \ge 1$ and the third inequality is because $\mathcal{Q}_k$ contains the smallest cut set for every $k$ consecutive sets. By Lemma 16, we have

$$\delta(Q'_i) \le \frac{1}{k} \cdot \sum_{Q_j \in \mathcal{Q}} \delta(Q_j) \le O(k^{-1/p} \cdot \alpha) \cdot \mathrm{OPT}\,.$$

Therefore, we have $\ell_p$-norm of the cut values of sets $Q'_1, Q'_2, \dots, Q'_k$ is at most

$$\left(\sum_{i=1}^{k} \delta(Q'_i)^p\right)^{1/p} \le \left(k \cdot O(k^{-1} \cdot \alpha^p) \cdot \mathrm{OPT}^p\right)^{1/p} = O(\alpha) \cdot \mathrm{OPT}\,.$$

Combining three parts, we get the conclusion. ◀

By Theorem 15, given a graph with $n$ vertices and $k$ terminals, our algorithm finds an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation for the $\ell_p$-Norm Multiway Cut with probability at least $3/4 - 1/k$. We can repeat this algorithm $O(\log 1/\varepsilon)$ times to find an $O(\log^{1/2} n \log^{1/2+1/p} k)$ approximation for the $\ell_p$-Norm Multiway Cut with probability at least $1 - \varepsilon$, which proves Theorem 1.

## References


1 Saba Ahmadi, Samir Khuller, and Barna Saha. Min-max correlation clustering via multicut. In *Proceedings of the Integer Programming and Combinatorial Optimization*, pages 13–26, 2019.

2 Haris Angelidakis, Yury Makarychev, and Pasin Manurangsi. An improved integrality gap for the Călinescu–Karloff–Rabani relaxation for multiway cut. In *Proceedings of Integer Programming and Combinatorial Optimization*, pages 39–50. Springer, 2017.

3 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Seffi, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, 43(2):872–904, 2014.

4 Kristóf Bérczi, Karthekeyan Chandrasekaran, Tamás Király, and Vivek Madan. Improving the integrality gap for multiway cut. *Mathematical Programming*, 183(1-2):171–193, 2020.

5 Niv Buchbinder, Joseph Naor, and Roy Schwartz. Simplex partitioning via exponential clocks and the multiway cut problem. In *Proceedings of the Symposium on Theory of Computing*, pages 535–544, 2013.

6 Niv Buchbinder, Roy Schwartz, and Baruch Weizman. Simplex transformations and the multiway cut problem. In *Proceedings of the Symposium on Discrete Algorithms*, pages 2400–2410, 2017.

7 Gruia Călinescu, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for multiway cut. In *Proceedings of the Symposium on Theory of Computing*, pages 48–52, 1998.

8 Karthekeyan Chandrasekaran and Weihang Wang. $\ell_p$-norm multiway cut. *Algorithmica*, 84(9):2667–2701, 2022.

9 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

10 Ari Freund and Howard Karloff. A lower bound of $8/(7 + \frac{1}{k-1})$ on the integrality ratio of the Călinescu–Karloff–Rabani relaxation for multiway cut. *Information Processing Letters*, 75(1-2):43–50, 2000.

11 David R Karger, Philip Klein, Cliff Stein, Mikkel Thorup, and Neal E Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proceedings of the Symposium on Theory of Computing*, pages 668–678, 1999.

12 Rajsekar Manokaran, Joseph Naor, Prasad Raghavendra, and Roy Schwartz. SDP gaps and UGC hardness for multiway cut, 0-extension, and metric labeling. In *Proceedings of the Symposium on Theory of Computing*, pages 11–20, 2008.

13 Ankit Sharma and Jan Vondrák. Multiway cut, pairwise realizable distributions, and descending thresholds. In *Proceedings of the Symposium on Theory of Computing*, pages 724–733, 2014.

14 Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *Proceedings of Approximation, Randomization, and Combinatorial Optimization*, pages 207–218, 2004.

# Polynomial-Time Approximation of Independent Set Parameterized by Treewidth

**Parinya Chalermsook** ✉
Aalto University, Finland

**Fedor Fomin** ✉
University of Bergen, Norway

**Thekla Hamm** ✉
Utrecht University, The Netherlands

**Tuukka Korhonen** ✉
University of Bergen, Norway

**Jesper Nederlof** ✉
Utrecht University, The Netherlands

**Ly Orgo** ✉
Aalto University, Finland

## Abstract

We prove the following result about approximating the maximum independent set in a graph. Informally, we show that any approximation algorithm with a "non-trivial" approximation ratio (as a function of the number of vertices of the input graph $G$) can be turned into an approximation algorithm achieving almost the same ratio, albeit as a function of the treewidth of $G$. More formally, we prove that for any function $f$, the existence of a polynomial time $(n/f(n))$-approximation algorithm yields the existence of a polynomial time $O(\mathsf{tw} \cdot \log f(\mathsf{tw})/f(\mathsf{tw}))$-approximation algorithm, where $n$ and $\mathsf{tw}$ denote the number of vertices and the width of a given tree decomposition of the input graph. By pipelining our result with the state-of-the-art $O(n \cdot (\log \log n)^2/\log^3 n)$-approximation algorithm by Feige (2004), this implies an $O(\mathsf{tw} \cdot (\log \log \mathsf{tw})^3/\log^3 \mathsf{tw})$-approximation algorithm.

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

**Keywords and phrases** Maximum Independent Set, Treewidth, Approximation Algorithms, Parameterized Approximation

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.33

**Funding** *Parinya Chalermsook*: Supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557).
*Fedor Fomin*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
*Thekla Hamm*: Supported by the Austrian Science Fund (FWF, project J4651-N).
*Tuukka Korhonen*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
*Jesper Nederlof*: Supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 853234).
*Ly Orgo*: Supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557).

**Acknowledgements** The research presented in this paper was initiated partially during the trimester on Discrete Optimization at Hausdorff Research Institute for Mathematics (HIM) in Bonn, Germany.





## 1 Introduction

An independent set of a graph is a subset of pairwise non-adjacent vertices. The Maximum Independent Set problem, which asks to find an independent set of maximum cardinality of a given input graph on $n$ vertices, has been among the most fundamental optimization problems that appeared in many research areas of computer science and has been a canonical problem of study in algorithms.

In the field of *approximation algorithms*, the problem is notoriously hard: It has no $O(n/2^{\log^{3/4} n})$-approximation algorithm running in polynomial time unless $NP$ can be solved in randomized quasi-polynomial time by the work of Khot and Ponnuswami [15] (building on earlier work by among others Håstad [14]). The best known polynomial time approximation algorithm is an $\tilde{O}(n/\log^3 n)$-approximation by Feige [9], which is almost twenty years old; here the $\tilde{O}$-notation hides factors polynomial in $\log\log n$.

Besides measuring the approximation ratio as a function of $n$, two other directions have been suggested in the literature. One of the directions is to measure the ratio as a function of the maximum degree $d$ of the input graph. The first improvement over the naive greedy $(d+1)$-approximation to $o(d)$ was given by Halldorsson and Radhakrishnan [12] in 1994. After this, several improvements to this approximation were made [1, 11, 13], culminating in the currently best $\tilde{O}(d/\log^{1.5} d)$-approximation by Bansal, Gupta, and Guruganesh [4] with an almost matching lower bound of $\Omega(d/\log^2 d)$ under the Unique Games Conjecture (UGC) by Austrin, Khot, and Safra [2]; here the $\tilde{O}$-notation hides factors polynomial in $\log\log d$.

Another direction is to measure the approximation ratio as a function of the *treewidth* of the input graph. Here, a simple greedy algorithm that is based on the fact that graph of treewidth $\mathsf{tw}$ are $\mathsf{tw}$-degenerate (see Lemma 2.5) achieves an approximation ratio of $(\mathsf{tw}+1)$. This was improved by Czumaj, Halldórsson, Lingas, and Nilsson [7] in 2005, who gave a $(\mathsf{tw}/\log n)$-approximation algorithm when a tree decomposition of width $\mathsf{tw}$ is given with the input graph. Their algorithm is quite elegant and follows easily from the observation that one can greedily partition the vertices of the graph into sets $V_1, \ldots, V_r$ such that the treewidth of $G[V_i]$ is at most $\mathsf{tw}/r$. Combined with dynamic programming for independent set on graphs of bounded treewidth, this gives a $2^{\mathsf{tw}/r} n^{O(1)}$ time $r$-approximation for any $r$, and therefore runs in polynomial time when we set $r = \mathsf{tw}/\log n$, resulting in the $(\mathsf{tw}/\log n)$-approximation algorithm.

Contrary to the degree-direction of approximating independent set, there has been no progress in the two other directions measuring the approximation ratio as a function on the number of vertices or the treewidth since the milestone results of Feige [9] and Czumaj et al. [7]. It is easy to show that one cannot improve the result of Czumaj et al. [7] to a polynomial time $(\mathsf{tw}/(f(\mathsf{tw})\log n))$-approximation for any diverging positive function $f$, assuming the Exponential Time Hypothesis (ETH). In particular, given an input graph $G$ on $n_0$ vertices we can create a graph $G'$ on $n = 2^{n_0/f(n_0)}$ vertices by adding $n - n_0$ vertices of degree 0. Then $G'$ has treewidth $n_0$ and the assumed algorithm is a $n^{O(1)} = 2^{o(n_0)}$-time $r$-approximation for $r = n_0/(f(n_0)\log n) = 1$, which violates the lower bound that Maximum Independent Set cannot be solved exactly in $2^{o(n_0)}$ time on graphs with $n_0$ vertices, assuming ETH (see e.g. for an equivalent lower bound for Vertex Cover [6, Theorem 14.6] ) .

This ETH lower bound naturally brings us to the question of what is the best approximation ratio in terms of treewidth only. In this paper, we essentially resolve this question by relating the approximation ratio parameterized by treewidth tightly to the approximation ratio parameterized by $n$.

Formally, as our main result we prove the following theorem:

▶ **Theorem 1.1.** *Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that there exists an $\frac{n}{f(n)}$-approximation algorithm for Maximum Independent Set, where $n$ is the number of vertices of the input graph*[1]*. Then there exists an $O\left(\frac{\mathsf{tw} \cdot \log f(\mathsf{tw})}{f(\mathsf{tw})}\right)$-approximation algorithm for Maximum Independent Set, where $\mathsf{tw}$ is the width of a given tree decomposition of the input graph.*

Let $\gamma(n)$ be the approximability function of Maximum Independent Set for $n$-vertex graph (i.e., the function for which $O(\gamma(n))$-approximation exists and $o(\gamma(n))$-approximation is hard). As mentioned before, the current state of the art has provided the lower and upper bounds $\gamma(n) = \Omega(n/2^{\log^{3/4} n})$ [15, 14] and $\gamma(n) = \tilde{O}(n/\log^3 n)$ respectively [9]. Similarly, one can consider Maximum Independent Set parameterized by $\mathsf{tw}$ and define $\tau(\mathsf{tw})$ as the approximability function of Maximum Independent Set on the setting when a tree decomposition of width $\mathsf{tw}$ is given. Our result implies that the approximability functions $\gamma$ and $\tau$ are essentially the same function, so this closes the treewidth-direction of Maximum Independent Set approximation.

We find this phenomenon rather surprising. For some other parameters, such relations do not hold, e.g., when we consider the degree parameter $d$ of the input graph, the approximability function of Maximum Independent Set is $\Omega(d/\log^2 d)$ assuming UGC [2], while the $\tilde{O}(n/\log^3 n)$-approximation of Feige [9] exists.

Combining Theorem 1.1 with the result of Feige [9], we obtain the following corollary.

▶ **Corollary 1.2.** *There exists an $O\left(\frac{\mathsf{tw} \cdot (\log\log \mathsf{tw})^3}{\log^3 \mathsf{tw}}\right)$-approximation algorithm for Maximum Independent Set, where $\mathsf{tw}$ is the width of a given tree decomposition of the input graph.*

This improves over the result of Czumaj et al. [7] when $\log^{1/3} n = o\left(\frac{\log \mathsf{tw}}{\log\log \mathsf{tw}}\right)$, i.e., when $\mathsf{tw}$ is larger than $\exp(\tilde{\Omega}(\log^{1/3} n))$. It is better than the algorithm of Feige [9] whenever $\mathsf{tw} = o(n/\log\log n)$, so overall it improves the state-of-the-art in the range of parameters

$$\exp(\tilde{\Omega}(\log^{1/3} n)) \leq \mathsf{tw} \leq o(n/\log\log n).$$

These results assume that the tree decomposition is given as part of the input. To remove this assumption, we can use the algorithm of Feige et al. [10] to $O(\sqrt{\log \mathsf{tw}})$-approximate treewidth. In particular, their algorithm combined with Corollary 1.2 yields the following corollary in the setting when a tree decomposition is not assumed as a part of the input.

▶ **Corollary 1.3.** *There exists an $O\left(\frac{\mathsf{tw} \cdot (\log\log \mathsf{tw})^3}{\log^{2.5} \mathsf{tw}}\right)$-approximation algorithm for Maximum Independent Set, where $\mathsf{tw}$ is the treewidth of the input graph.*

## Techniques

On a high level, our technique behind Theorem 1.1 is as follows: First we delete a set of vertices of size at most $\mathsf{OPT}/2$ from the graph so that each of the remaining components can be partitioned into subinstances with pathwidth at most $\mathsf{tw}$ and subinstances with tree decompositions of width $O(\mathsf{tw})$ and depth $O(\log f(\mathsf{tw}))$. For the subinstances of small pathwidth, we partition the vertices into $O(\log f(\mathsf{tw}))$ levels based on in how many bags of the path decomposition they occur. Similarly, for the subinstances with $O(\log f(\mathsf{tw}))$-depth tree decompositions, we partition the vertices in levels based on the depth of the highest bag of the tree decomposition they occur in. In both subinstances we argue that all vertices of

[1] We make mild assumptions on the properties of $f$, which are detailed in Section 2. Any "reasonable" function $f$ satisfies these assumptions.

all but one level can be removed, in order to make the vertices in the remaining level behave well in the decomposition, after which the remaining level can be chopped into components of size roughly $O(\mathsf{tw})$ such that the size of maximum independent again does not decrease significantly.

Although some aspects of our approach are natural, we are not aware of arguments modifying the tree decomposition as we did here in the previous literature; we expect these arguments may have more applications for designing approximation algorithms for other $NP$-hard problem parameterized by treewidth similar to Theorem 1.1.

### Organization

The paper is organized as follows. We give preliminaries in Section 2. A major ingredient of Theorem 1.1 will be an approximation algorithm for Maximum Independent Set parameterized by pathwidth, which we will be presented in Section 3. Then, the approximation algorithm for Maximum Independent Set parameterized by treewidth will be presented in Section 4. This will use the pathwidth case as a black box. We then conclude and present open problems in Section 5.

## 2 Preliminaries

### Basic notation

We refer to [8] for standard graph terminology. We use the standard notation – $\alpha(G)$ – to denote the independence number, i.e., the size of a maximum independendent set, of graph $G$. Throughout, for a natural number $i$ we denote the set $\{1, \ldots, i\}$ by $[i]$, and for two natural numbers $i \leq j$ we denote the set $\{i, i+1, \ldots, j\}$ by $[i, j]$. We use log to denote the base-2 logarithm.

### Tree decompositions

Given a graph $G$, a tree decomposition of $G$ consists of a tree $T$, where each node $t \in V(T)$ is associated with a subset $B_t \subseteq V(G)$ of vertices called a bag, such that

1. $\bigcup_{t \in V(T)} B_t = V(G)$
2. For every edge $uv \in E(G)$, there must be some node $t$ such that $\{u, v\} \subseteq B_t$.
3. For every vertex $v \in V(G)$, the bags $\{t : v \in B_t\}$ are connected in $T$.

The **width** of a tree decomposition is $\max_{t \in V(T)} |B_t| - 1$. The **treewidth** of $G$ (denoted by $\mathsf{tw}(G)$) is the minimum number $k$, such that $G$ has a tree decomposition of width $k$. When the input graph is clear from the context, we simply write $\mathsf{tw}$ to denote the treewidth of $G$.

A rooted tree decomposition is a tree decomposition where one node is assigned to be the root of the tree $T$. We use standard rooted-tree definitions when talking about rooted tree decomposition. The depth of a rooted tree decomposition is the depth of the tree $T$, i.e., the length of the longest root-leaf path.

A rooted tree decomposition $T$ is called **nice** if it satisfies that

- Every node of $T$ has at most 2 children.
- If a node $t$ has two children $t'$ and $t''$, then $t$ is called a join node and $B_t = B_{t'} = B_{t''}$.
- If a node $t$ has one child $t'$, then either:
  1. $B_t \subset B_{t'}$ and $|B_{t'}| = |B_t| + 1$, in which case $t$ is a forget node, or
  2. $B_{t'} \subset B_t$ and $|B_t| = |B_{t'}| + 1$, in which case $t$ is an introduce node.
- If a node $t$ has no children we call it a leaf node.

It is well-known that any tree decomposition can be turned into a nice tree decomposition.

▶ **Lemma 2.1** ([16]). *For every graph $G$ on $n$ vertices, given a tree decomposition $T'$ of width $\omega$, there is a nice tree decomposition $T$ with at most $4 \cdot n$ nodes and width $\omega$ that can be computed in polynomial time.*

It is possible to also assume the following additional property without loss of generality.

▶ **Lemma 2.2.** *Given a tree decomposition $T'$ of width $\omega$, there exists a nice tree decomposition of width $\omega$ and at most $4n$ nodes, that can be computed in polynomial time, such that for each leaf node $t \in V(T)$, there exists a vertex $v \in B_t$ that appears in exactly one bag, i.e., the bag $B_t$ itself.*

**Proof.** We use Lemma 2.1 to compute a nice tree decomposition $T$. If there exists a leaf node $t \in V(T)$ that does not contain such a vertex, we delete $t$ from $T$. Notice that all the properties of a tree decomposition continue to hold after such a deletion. However, if after this deletion the former parent $s$ of $t$ in $T$ is not a leaf, $s$ was a join node which now has a child with the same bag as $s$ which violates niceness. To repair this we can simply contract the edge between $s$ and its remaining child in $T$. It is straightforward to verify that after this $T$ remains nice. We can iterate the above, strictly decreasing the number of nodes of $T$, until $T$ has the desired property. ◀

We will use the following well-known lemma of Bodlaender and Hagerup [5] to turn a tree decomposition into a logarithmic-depth tree decomposition, while increasing the width only by a factor of three.

▶ **Lemma 2.3** ([5, Lemma 2.2]). *Given a tree decomposition of a graph $G$ of width $\omega$ and having $\gamma$ nodes, we can compute in polynomial time a rooted tree decomposition of $G$ of depth $O(\log \gamma)$ and width at most $3\omega + 2$.*

### Path decompositions

A path decomposition is a tree decomposition where the tree $T$ is a path. The **pathwidth** of $G$ is the minimum number $k$, such that $G$ has a path decomposition of width $k$. It is denoted by $\mathsf{pw}(G)$. A nice path decomposition is a nice tree decomposition where $T$ is a path, and the root is assigned to a degree-1 node, i.e., at one end of the path. Note that there are no join-nodes in a nice path decomposition.

We observe that any path decomposition can be turned into a nice path decomposition with $2n$ nodes.

▶ **Lemma 2.4.** *For every graph $G$ on $n$ vertices, given a path decomposition $P'$ of width $\omega$, there is a nice path decomposition $P$ with $2n$ nodes and width $\omega$, that can be computed in polynomial time.*

**Proof.** By introducing vertices one at a time and forgetting vertices one at a time we obtain a nice path decomposition where the bag of the first node is empty, the bag of the last node is empty, and on each edge exactly one vertex is either introduced or forgotten, and therefore the path decomposition has exactly $2n$ edges and $2n + 1$ nodes. We can remove the first bag that is empty to get a path decomposition with exactly $2n$ nodes. ◀

### Maximum independent set approximation

Given a function $r$ that maps graphs to numbers greater than 1, an $r$-approximation algorithm for Maximum Independent Set takes as input a graph $G$ and outputs in polynomial time an independent set in $G$ of size at least $\frac{\alpha(G)}{r(G)}$. We usually denote any occurrence of $|V(G)|$ in $r$ by $n$.

Let us now detail our assumptions on the function $f$ in Theorem 1.1. We assume that the approximation ratio $n/f(n)$ of the given approximation algorithm is a non-decreasing function on $n$. This assumption is reasonable because if $n/f(n)$ would be decreasing at some point, we could improve the approximation ratio by adding universal vertices to the graph; note that adding universal vertices does not change the optimal solution, but increases $n$. This also implies that the function $f(n)$ grows at most linearly in $n$. We assume that for arbitrary fixed constant $c \geq 1$, it holds that $f(c \cdot n) \in O(f(n))$. We also assume that the function $f$ can not decrease too much when $n$ grows, in particular, we assume that for arbitrary fixed constant $c \geq 1$ it holds that $f(c \cdot n) \in \Omega(f(n))$.

Moreover, we will use a basic result about finding independent sets whose size depends on the treewidth of the graph. Recall that a graph $G$ is $d$-degenerate if there is always a vertex of degree at most $d$ in any induced subgraph of $G$. It is known that every graph $G$ is $\mathsf{tw}(G)$-degenerate: Simply consider the vertex that is contained at a leaf bag and no other bag of a tree decomposition $T$ of any induced subgraph of $G$ as given by Lemma 2.2. This vertex has degree at most $\mathsf{tw}(G)$. Therefore, we obtain a following trivial algorithm for approximating Maximum Independent Set parameterized by treewidth.

▶ **Lemma 2.5.** *There is a polynomial time algorithm that given a graph $G$ on $n$ vertices finds an independent set of size at least $n/(\mathsf{tw}(G)+1)$.*

**Proof.** Iteratively assign a vertex of minimum degree to the independent set and delete its neighbors. By the aforementioned degeneracy argument, at each iteration at most $\mathsf{tw}(G)+1$ vertices are deleted, so the number of iterations and the size of the found independent set is at least $n/(\mathsf{tw}(G)+1)$. ◀

Note that the algorithm of Lemma 2.5 does not need a tree decomposition as an input.

## 3 Approximation parameterized by pathwidth

In this section, we prove a version of Theorem 1.1 where instead of a tree decomposition, the input graph is given together with a path decomposition. This will be an important ingredient for proving Theorem 1.1. In particular, this section is devoted to the proof of the following lemma.

▶ **Lemma 3.1.** *Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that there exists an $\frac{n}{f(n)}$-approximation algorithm for Maximum Independent Set, where $n$ is the number of vertices of the input graph, and $f$ satisfies the assumptions outlined in Section 2. Then there exists an $O\left(\frac{\mathsf{pw}\cdot\log f(\mathsf{pw})}{f(\mathsf{pw})}\right)$-approximation algorithm for Maximum Independent Set, where $\mathsf{pw}$ is the width of a given path decomposition of the input graph.*

Throughout this section we will use $G$ to denote the input graph and $\mathsf{pw}$ to denote the width of the given path decomposition of $G$. We denote by $k = \mathsf{pw}+1$ the maximum size of a bag in the given decomposition. Note that by our assumptions on the function $f$, it holds that $f(k) = \Theta(f(\mathsf{pw}))$.

Let us denote $\mathsf{OPT} = \alpha(G)$. If $\mathsf{OPT} < \frac{n}{f(k)}$, then Lemma 2.5 gives us a solution of size at least

$$\frac{n}{\mathsf{tw}(G)+1} \geq \mathsf{OPT} \cdot \frac{f(k)}{\mathsf{tw}(G)+1},$$

i.e., an $O(\mathsf{tw}(G)/f(k))$-approximation, which would give the desired result by the facts that $\mathsf{tw}(G) \leq \mathsf{pw}$ and $f(k) = \Omega(f(\mathsf{pw}))$. Therefore, in the rest of this section we will assume that $\mathsf{OPT} \geq \frac{n}{f(k)}$.

Let $P$ be the given path decomposition of $G$. By Lemma 2.4, we can assume without loss of generality that $P$ is a nice path decomposition and has exactly $2n$ bags, which we will denote by $B_1, \ldots, B_{2n}$ in the order they occur in the path. For each $v \in V(G)$, we define the **length** of $v$ to be the number of bags in $P$ that contain $v$, and denote the length of $v$ by $\ell(v)$. In particular,

$$\ell(v) = |\{i \in [1, 2n] \colon v \in B_i\}|.$$

Then, we partition $V(G)$ into $2 + \lceil \log f(k) \rceil$ sets based on the lengths of the vertices:

$$\begin{aligned} V_0 &= \{v \colon \ell(v) < 2k\} \\ V_i &= \{v \colon \ell(v) \in [k \cdot 2^i, k \cdot 2^{i+1})\}, && 1 \leq i \leq \lceil \log f(k) \rceil \\ V' &= \{v \colon \ell(v) \geq 4k \cdot 2^{\lceil \log f(k) \rceil}\} \end{aligned}$$

Note that $(V_0, V_1, \ldots, V_{\lceil \log f(k) \rceil}, V')$ is indeed a partition of $V(G)$. We first show that the set $V'$, which consists of the longest vertices, can only contribute to at most half of the optimal solution.

▶ **Lemma 3.2.** *It holds that* $|V'| \leq \mathsf{OPT}/2$.

**Proof.** First, notice that $\sum_{v \in V(G)} \ell(v) \leq 2nk$. This is because $P$ has $2n$ bags, each vertex appears in $\ell(v)$ bags of $P$, and each bag of $P$ can have at most $k$ vertices appearing in it. Now, because for vertices $v \in V'$ we have $\ell(v) \geq 4k \cdot 2^{\lceil \log f(k) \rceil} \geq 4k \cdot f(k)$ the vertices in $V'$ contribute at least $\sum_{v \in V'} \ell(v) \geq 4k \cdot f(k) \cdot |V'|$ to the sum. Therefore, it holds that

$$|V'| \leq \frac{2nk}{4k \cdot f(k)} \leq \frac{n}{2 \cdot f(k)} \leq \mathsf{OPT}/2,$$

as desired. ◀

Lemma 3.2 implies that at least half of any maximum independent set in $G$ must be in the subgraph $G[V_0 \cup V_1 \cup \ldots \cup V_{\lceil \log f(k) \rceil}]$. In the rest of this section, we will focus on the following lemma.

▶ **Lemma 3.3.** *For each* $i \in [0, \lceil \log f(k) \rceil]$*, there is a* $O(k/f(k))$*-approximation algorithm for Maximum Independent Set in* $G[V_i]$.

It is easy to see how Lemma 3.3 implies Lemma 3.1. For each such $G[V_i]$, we invoke Lemma 3.3 to obtain a $O(k/f(k))$-approximate solution $S_i \subseteq V_i$. Our algorithm returns the set $S_i$ with the largest cardinality. Since there are at most $O(\log f(k))$ such sets, by Lemma 3.2 there must be some integer $i^*$ for which $\alpha(G[V_{i^*}]) \geq \Omega(\mathsf{OPT}/\log f(k))$. Therefore, the returned set must have size at least

$$\frac{\Omega(\mathsf{OPT}/\log f(k))}{O(k/f(k))} = \mathsf{OPT} \cdot \Omega\left(\frac{f(k)}{k \cdot \log f(k)}\right) = \mathsf{OPT} \cdot \Omega\left(\frac{f(\mathsf{pw})}{\mathsf{pw} \cdot \log f(\mathsf{pw})}\right).$$

Therefore, to finish the proof of Lemma 3.1, it remains to prove Lemma 3.3.

**Proof of Lemma 3.3.** Recall that the bags of $P$ are denoted by $B_1, B_2, \ldots, B_{2n}$ where $B_h$ is the $h$-th bag in the order from left to right. Let $L = \max_{v \in V_i} \ell(v)$ denote the maximum length of a vertex $v \in V_i$. Recall that by our definition of $V_i$, it holds that if $i = 0$, then $L < 2k$, and if $i > 0$, then all vertices in $V_i$ have length between $L/2$ and $L$. We partition the set $V_i$ into sets $X_r$ and $Y_r$ as follows.

- For each $r \in [1, \lfloor 2n/(2L) \rfloor]$, we define $X_r = B_{2Lr} \cap V_i$. These sets contain the vertices of $V_i$ that appear in bags $B_{2L}, B_{4L}, \ldots$ and the vertices in $B_{2Lr}$ can never occur in the same bag as the vertices in $B_{2Lr'}$, for any $r \neq r'$, since all vertices in $V_i$ have length at most $L$. Let $X = \bigcup_r X_r$.
- Denote the remaining vertices by $Y = V_i \setminus X$. We further partition $Y$ into sets $Y_r$ for $r \in [1, \lceil 2n/(2L) \rceil]$, where $Y_r$ contains the vertices $v \in Y$ that occur only in the bags $B_j$ in the interval $j \in [2L(r-1)+1, 2Lr-1]$.

It follows from definitions that $X \cup Y = V_i$. See Figure 1 for an illustration.

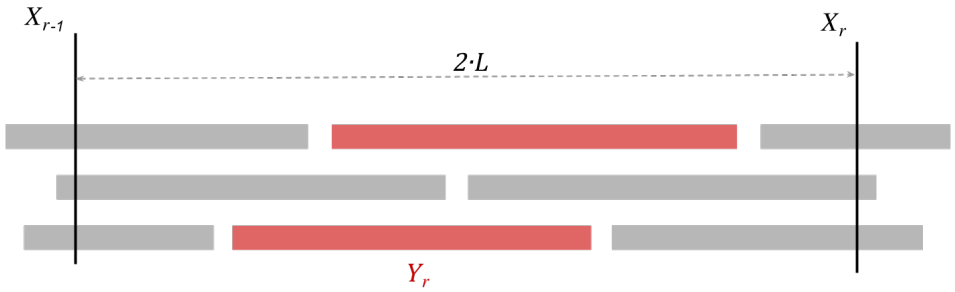



**Figure 1** $X_r$ is the set of vertices in $V_i$ that are in the $(2Lr)$-th bag. $Y_r$ (in red) is the set of vertices in $V_i$ that start after $X_{r-1}$ and end before $X_r$.

We prove the following claim.

▷ Claim 3.4. For all $r \in \mathbb{N}$, both sets $X_r$ and $Y_r$ have size at most $4k$.

Proof of claim. For the set $X_r$, there is nothing to prove since each bag contains at most $k$ vertices. Let us consider the set $Y_r$. First, we observe that because vertices of $Y_r$ occur only in the bags $B_{2L(r-1)+1}, \ldots, B_{2Lr-1}$, we have that

$$\sum_{v \in Y_r} \ell(v) \leq k \cdot 2L, \tag{1}$$

by the argument that each bag can contribute to the length of at most $k$ vertices. Then, we consider two cases: $i > 0$ and $i = 0$.

In the case when $i > 0$, we know that each vertex $v \in Y_r$ has length at least $\ell(v) \geq L/2$. Together with Equation (1), this implies that $|Y_r| \leq 4k$.

In the case when $i = 0$, we have $L \leq 2k$, but we do not have the lower bound on the length of vertices in $V_0$. In this case, we use the property that $P$ is a nice path decomposition of $G$. We know that the paths of vertices in $Y_r$ appear only in the bags $B_{2L(r-1)+1}, \ldots, B_{2Lr-1}$ of $P$. There are $2L - 1$ such bags, and because $P$ is nice, each bag either introduces a single vertex in $Y_r \cup X_r$ or forgets a single vertex in $X_{r-1} \cup Y_r$. Since all vertices of $Y_r$ must be introduced in these bags, but there are only $2L - 1$ such bags, this implies that $|Y_r| \leq 2L - 1 \leq 4k$. ◁

Finally, notice that because there are no bags that contain vertices from both $Y_r$ and $Y_{r'}$ for $r \neq r'$, there are no edges between $Y_r$ and $Y_{r'}$ for $r \neq r'$. Also, since $X_{r-1}$ and $X_r$ have $2L-1$ bags between them and the maximum length of a vertex is $L$, it follows that no vertex from $X_{r-1}$ occurs in a bag together with a vertex in $X_r$, and therefore there are no edges between $X_r$ and $X_{r'}$ for $r \neq r'$. Therefore, a union of independent sets in $Y_1, \ldots, Y_{\lceil 2n/(2L) \rceil}$ is an independent set in $Y$, and a union of independent sets in $X_1, \ldots, X_{\lfloor 2n/(2L) \rfloor}$ is an independent set in $X$.

As each graph $G[X_r]$ and $G[Y_r]$ has at most $4k$ vertices, we use the given $\frac{n}{f(n)}$-approximation algorithm to $\frac{4k}{f(4k)}$-approximate maximum independent set in all of the graphs $G[X_r]$ and $G[Y_r]$. We denote by $X^*$ the union of the results in the graphs $G[X_r]$ and by $Y^*$ the union of the results in the graphs $G[Y_r]$. Note that by previous arguments, $\alpha(G[X]) = \sum_r \alpha(G[X_r])$ and $\alpha(G[Y]) = \sum_r \alpha(G[Y_r])$, and therefore $X^*$ is a $\frac{4k}{f(4k)}$-approximation for independent set in $G[X]$ and $Y^*$ is a $\frac{4k}{f(4k)}$-approximation for independent set in $G[Y]$. Now, we observe that because $V_i = X \cup Y$, either $\alpha(G[X]) \geq \alpha(G[V_i])/2$ or $\alpha(G[Y]) \geq \alpha(G[V_i])/2$, and therefore the larger of $X^*$ and $Y^*$ is a $\frac{8k}{f(4k)}$-approximation for independent set in $G[V_i]$. Note that $\frac{8k}{f(4k)} = O(k/f(k))$, which is the desired approximation ratio. ◀

## 4 Approximation parameterized by treewidth

In this section, we finish the proof of Theorem 1.1. For the convenience of the reader, let us re-state Theorem 1.1 here.

▶ **Theorem 1.1.** *Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that there exists an $\frac{n}{f(n)}$-approximation algorithm for Maximum Independent Set, where $n$ is the number of vertices of the input graph*[2]*. Then there exists an $O\left(\frac{\mathsf{tw} \cdot \log f(\mathsf{tw})}{f(\mathsf{tw})}\right)$-approximation algorithm for Maximum Independent Set, where $\mathsf{tw}$ is the width of a given tree decomposition of the input graph.*

Throughout this section we will use $G$ to denote the input graph and $\mathsf{tw}$ to denote the width of the given tree decomposition of $G$. We denote by $k = \mathsf{tw} + 1$ the maximum size of a bag in the given tree decomposition. Recall that $f(k) = \Theta(f(\mathsf{tw}))$.

Let $T$ be the given tree decomposition of $G$. By Lemma 2.2 we assume that $T$ is nice, and moreover that for each leaf node $t$ of $T$ there exists a vertex $v \in B_t$ that occurs only in the bag $B_t$. Let $\mathsf{OPT}$ denote the size of a maximum independent set in $G$. Similarly to the pathwidth case in Section 3, by using Lemma 2.5 we can assume in the rest of this section that $\mathsf{OPT} \geq \frac{n}{f(k)}$.

Let $\mathcal{L} \subseteq V(T)$ be the set of all leaf nodes of $T$. If the number of leaf nodes is at least $|\mathcal{L}| \geq \frac{\mathsf{OPT} \cdot f(k)}{k}$, then the unique vertices in these leaf bags already give us an independent set with the desired approximation factor. Therefore, in the rest of this section we will also assume that $|\mathcal{L}| < \frac{\mathsf{OPT} \cdot f(k)}{k}$. With this assumption, we can invoke the following lemma with $\ell = 2f(k)$.

▶ **Lemma 4.1.** *There exists a set $X \subseteq V(G)$ of size $|X| \leq k \cdot \frac{|\mathcal{L}|}{\ell}$ such that for each connected component of $G - X$ there is a rooted tree decomposition of width at most $k-1$ that has at most $\ell$ leaf nodes. Such a set $X$ and the tree decompositions of the components can be computed in polynomial time.*

[2] We make mild assumptions on the properties of $f$, which are detailed in Section 2. Any "reasonable" function $f$ satisfies these assumptions.

**Proof.** We prove the lemma constructively starting with $X = \emptyset$ and the tree decomposition $T$ of the entire graph. We also maintain a set of tree decompositions $\mathcal{C}$ of connected components of $G - X$. We iteratively remove vertices from the graph $G$ based on the structure of $T$ as follows.

Initially, we define the set of tree decompositions we will return as $\mathcal{C} = \emptyset$, and we initially assign $T' = T$ as the tree decomposition from which we will "chop off" pieces with at most $\ell$ leaf nodes into $\mathcal{C}$. As long as $T'$ has more than $\ell$ leaves, let $t^*$ be a node of $T'$ such that there are at least $\ell$ leaf nodes in the subtree $T^*$ of $T'$ rooted at $t^*$ and no descendant of $t^*$ has the same property. We add the vertices of $B_{t^*}$ to $X$ and delete them from the graph and all bags of $T'$. This separates the vertices in the bags in $T^*$ from the vertices in the bags of the rest of $T'$ and since no descendant of $t^*$ had more than $\ell$ leaves, all connected components of $T^* - t^*$ have at most $\ell$ leaves.

We remove $T^*$ from $T'$, and add all connected components of $T^* - t^*$ into $\mathcal{C}$. This completes the iteration. When the process stops, vertices in $G$ are either deleted (because they belonged to $B_{t^*}$ in some iteration) or appear in some tree decomposition that was added to $\mathcal{C}$.

By construction, each connected component of $G - X$ has a tree decomposition that is given by a connected component in $\mathcal{C}$. Each of these has fewer than $\ell$ leaves and did not increase in width compared to $T$.

With these observations, the following claim finishes the proof of the lemma.

▷ Claim 4.2. It holds that $|X| \leq k \cdot \frac{|\mathcal{C}|}{\ell}$.

Proof of claim. In each iteration of the algorithm, the number of leaves of $T'$ decreases by at least $\ell$, because $T^*$ has more than $\ell$ leaves. Hence, this process terminates after at most $\frac{|\mathcal{C}|}{\ell}$ iterations. Each such iteration adds a subset of a bag of $T$ to $X$ (which contains at most $k$ vertices). Therefore, the total number of deleted vertices is at most $k \cdot \frac{|\mathcal{C}|}{\ell}$. ◁

◀

We then assume to have $X$ as in the statement of Lemma 4.1 with $\ell = 2 \cdot f(k)$, and for each connected component $C$ of $G - X$ a tree decomposition $T^C$ of width at most $k - 1$ with at most $2f(k)$ leaves. For a connected component $C$ of $G - X$, let $S_C$ denote a fixed maximum independent set in $C$. Since $|X| \leq k \cdot \frac{|\mathcal{C}|}{2f(k)} \leq \mathsf{OPT}/2$, we know that the sum of $|S_C|$ over all connected components $C$ of $G - X$ is at least $\mathsf{OPT}/2$.

We can distinguish two cases for a single connected component $C$ of $G - X$ based on whether a majority of $S_C$ appears in bags of nodes of degree at least 3 in $T^C$ or not. Formally, let $Q$ denote the set of vertices that appear in the bags of nodes of degree at least 3 in $T^C$, i.e., $Q = \bigcup_{t \text{ has degree} > 2 \text{ in } T^C} B_t$. For each component $C$ one of the following two alternatives holds:

1. $|S_C \setminus Q| > |S_C|/2$, or
2. $|S_C \cap Q| \geq |S_C|/2$.

For handling the first case we can observe an easy pathwidth bound for $C - Q$, which allows us to apply Lemma 3.1.

▶ **Lemma 4.3.** *A path decomposition of $C - Q$ of width at most $k - 1$ can be computed in polynomial time given the tree decomposition $T^C$.*

**Proof.** A path decomposition witnessing this can easily be obtained from $T^C$ by deleting all nodes with degree at least 3 as well as vertices in their bags from the decomposition, resulting in a disjoint union of paths all of whose bags are of size at most $k$. These paths can be concatenated in arbitrary order. ◀

For the second case we next give a lemma that splits up each $C[Q]$ into $O(\log f(k))$ many disjoint subgraphs in which every connected component has at most $O(k)$ vertices.

▶ **Lemma 4.4.** *$C[Q]$ can be divided into $\ell \leq O(\log f(k))$ subgraphs $H_1, \ldots, H_\ell$, such that $V(C[Q]) = \dot{\bigcup}_{i\in[\ell]} V(H_i)$, and for any $i \in [\ell]$, each connected component of $H_i$ has at most $6k$ vertices. Such $H_1, \ldots, H_\ell$ can be computed in polynomial time.*

**Proof.** Consider the tree decomposition $T^C$ of $C$ obtained according to Lemma 4.1. In particular, such a tree has at most $2f(k)$ leaves and therefore at most $2f(k) - 1$ nodes with degree at least 3.

We can replace each path between two nodes $u$ and $v$ of degree at least 3 in $T^C$ by two edges incident to a shared new node whose bag consists of the union $B_u \cup B_v$ of the bags of $u$ and $v$. In this way we obtain a tree decomposition of $C[Q]$ with at most $4 \cdot f(k)$ nodes and width at most $2k - 1$.

For this tree decomposition we invoke Lemma 2.3 to obtain a tree decomposition $T^J$ of $C[Q]$ with width at most $6k - 1$ and depth $\ell \in O(\log f(k))$. Now, we partition the vertices in $C[Q]$ into $H_1, \ldots, H_\ell$ where $H_i$ contains all vertices $v$ such that the distance between the root of $T^J$ and the highest bag in which $v$ appears is exactly $i - 1$.

By definition all $V(H_i)$ are pairwise disjoint and because $T^J$ is a tree decomposition of $C[Q]$, the union of all $V(H_i)$ covers $V(C[Q])$. Moreover each connected component of any $H_i$ is by construction a subset of some bag of $T_J$ and thus has at most $6k$ vertices as desired. ◀

With the previous lemmas in hand, we are now ready to finish the proof of Theorem 1.1 as follows.

**Proof of Theorem 1.1.** We begin by invoking Lemma 4.1. Let $\mathcal{C}$ be the set of connected components in $G - X$.

For the next few paragraphs consider an arbitrary but fixed single connected component $C \in \mathcal{C}$. We first use Lemma 4.3 to invoke Lemma 3.1 on $C - Q$ to obtain an independent set $S_C^J$ in $C - Q$ of size at least $\Omega\left(\frac{f(k)}{k \cdot \log f(k)}\right) \cdot \alpha(C - Q)$.

Independently we invoke Lemma 4.4 and on each of the returned graphs $H_i$ the assumed $\frac{n}{f(n)}$-approximation for $n$-vertex graphs on each of its connected components. Due to their small component size for each $H_i$ this results in an independent set $S_{H_i}$ of size at least $\Omega\left(\frac{f(k)}{k}\right) \cdot \alpha(H_i)$. Because the graphs $H_i$ vertex-partition $C[Q]$ and there are only $O(\log f(k))$ many $H_i$, returning an $S_{H_i}$ with maximum size yields an $O(k \log f(k)/f(k))$-approximate solution for Maximum Independent Set on $C[Q]$.

We know that either

1. $\alpha(C - Q) \geq \alpha(C)/2$, or
2. $\alpha(C[Q]) \geq \alpha(C)/2$.

Overall this implies that returning the larger of $S_C^J$ and the maximum-size $S_{H_i}$ yields an $O\left(\frac{k \log f(k)}{f(k)}\right)$-approximate solution for Maximum Independent Set on $C$. We denote the returned independent set by $S_C$.

Our final output is the union of all $S_C$. Because all $C$ are pairwise independent, the union of $S_C$ is an independent set in $G$. Moreover, because $\sum_{C\in\mathcal{C}} \alpha(C) \geq \mathsf{OPT}/2$ and because of the above approximation guarantee for each $S_C$, we obtain the overall desired approximation guarantee. ◀

## 5 Conclusion and open problems

In this paper we essentially settled the polynomial time approximability of Maximum Independent Set when parameterized by treewidth. The most relevant open problem is to extend our approach to give the improved time-approximation tradeoff result in Czumaj et al. [7]. The current best known algorithm gives an $r$-approximation in $2^{\mathsf{tw}/r}n^{O(1)}$ time. With fine-tuning of the parameters and using the recent exponential-time approximation result of Bansal et. al. [3], we believe our techniques could give an improved running time of $2^{o(\mathsf{tw}/r)}$ when $r$ is sufficiently high, e.g., $r = \log^{\Omega(1)} \mathsf{tw}$.

For us, the most interesting question is perhaps when $r$ is tiny. Can we get a 2-approximation algorithm that runs in time $2^{(1/2-\epsilon)\mathsf{tw}}n^{O(1)}$? Can we prove some concrete lower bound in this regime? While the Gap-ETH lower bound $2^{\mathsf{tw}/\mathsf{poly}(r)}$ (for sufficiently large $r$) is immediate from [3], such techniques do not rule out anything when $r$ is a small constant.

A different possible direction for future research would be to formulate approximation algorithms in terms of treewidth only for the more general Maximum Weight Induced Subgraph problem studied by Czumaj et al. [7].

## References


1 Noga Alon and Nabil Kahalé. Approximating the independence number via the theta-function. *Math. Program.*, 80:253–264, 1998. doi:10.1007/BF01581168.

2 Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of vertex cover and independent set in bounded degree graphs. *Theory Comput.*, 7(1):27–43, 2011. doi:10.4086/toc.2011.v007a003.

3 Nikhil Bansal, Parinya Chalermsook, Bundit Laekhanukit, Danupon Nanongkai, and Jesper Nederlof. New tools and connections for exponential-time approximation. *Algorithmica*, 81:3993–4009, 2019.

4 Nikhil Bansal, Anupam Gupta, and Guru Guruganesh. On the lovász theta function for independent sets in sparse graphs. *SIAM J. Comput.*, 47(3):1039–1055, 2018. doi:10.1137/15M1051002.

5 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

7 Artur Czumaj, Magnús M Halldórsson, Andrzej Lingas, and Johan Nilsson. Approximation algorithms for optimization problems in graphs with superlogarithmic treewidth. *Information processing letters*, 94(2):49–53, 2005.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Uriel Feige. Approximating maximum clique by removing subgraphs. *SIAM Journal on Discrete Mathematics*, 18(2):219–225, 2004.

10 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008. doi:10.1137/05064299X.

11 Magnús M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *J. Graph Algorithms Appl.*, 4(1):1–16, 2000. doi:10.7155/jgaa.00020.

12 Magnús M. Halldórsson and Jaikumar Radhakrishnan. Improved approximations of independent sets in bounded-degree graphs via subgraph removal. *Nord. J. Comput.*, 1(4):475–492, 1994.

13 Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 329–337. ACM/SIAM, 2000. URL: `http://dl.acm.org/citation.cfm?id=338219.338269`.

14 Johan Hastad. Clique is hard to approximate within n/sup 1-/spl epsiv. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 627–636. IEEE, 1996.

15 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I 33*, pages 226–237. Springer, 2006.

16 Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

# Faster Local Motif Clustering via Maximum Flows

**Adil Chhabra**
Heidelberg University, Germany

**Marcelo Fonseca Faraj**
Heidelberg University, Germany

**Christian Schulz**
Heidelberg University, Germany

## Abstract

Local clustering aims to identify a cluster within a given graph that includes a designated seed node or a significant portion of a group of seed nodes. This cluster should be well-characterized, i.e., it has a high number of internal edges and a low number of external edges. In this work, we propose `SOCIAL`, a novel algorithm for local motif clustering which optimizes for motif conductance based on a local hypergraph model representation of the problem and an adapted version of the max-flow quotient-cut improvement algorithm (`MQI`). In our experiments with the triangle motif, `SOCIAL` produces local clusters with an average motif conductance 1.7% lower than the state-of-the-art, while being up to multiple orders of magnitude faster.

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

**Keywords and phrases** local motif clustering, motif conductance, maximum flows, max-flow quotient-cut improvement

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.34

**Related Version** *Full Version*: `https://arxiv.org/pdf/2301.07145.pdf`

**Supplementary Material** *Software (Code)*:
`https://github.com/LocalClustering/HeidelbergMotifClustering`

**Funding** DFG grant SCHU 2567/5-1.

## 1 Introduction

Graphs are a fundamental tool for representing complex systems and relationships in a wide range of contexts. They can be used to model everything from data dependencies and social networks to web links and email interactions. With the massive expansion of data in recent years, many real-world graphs have grown to enormous sizes, making it challenging to analyze them. In particular, many applications only require analyzing a small, localized portion of a graph rather than the entire graph, which is the case for community-detection on Web [12] and social [21] networks as well as structure-discovery in bioinformatics [47] networks, among others. Those real-world applications are usually preceded by or modeled as a *local clustering* problem. Local clustering aims at identifying a specific cluster within a given graph that includes a designated seed node or a portion of a group of seed nodes, and is *well-characterized*, i.e., it consists of many internal edges and few external edges. More specifically, the quality of a community can be quantified by specific metrics such as *conductance* [22]. Since minimizing conductance is NP-hard [48], approximate and heuristic approaches are used in practice. Given the nature and scale of the problem, these approaches should ideally require time and memory dependent only on the size of the found cluster.

The local clustering problem has been investigated both theoretically [1] and experimentally [29], and has been solved using a wide variety of techniques, including statistical [9, 24], numerical [30, 32], and combinatorial [35, 15] methods. While traditional approaches to local


clustering typically consider the edge distribution when evaluating the quality of a local community, novel methods [49, 50, 33, 34, 7] have shifted focus to finding local communities based on the distribution of *motifs*, higher-order structures within the graph. These works provide empirical evidence that this approach, which can be called *local motif clustering*, is effective at detecting high-quality local communities. Nevertheless, since this local clustering perspective is relatively new, there are still many opportunities to improve upon current approaches and discover more efficient algorithms for finding high-quality solutions.

**Contribution.** In this work, we propose a novel algorithm for local motif clustering which optimizes for motif conductance by combining the strongly local hypergraph model from Chhabra et al. [7] with an adapted version of the fast and effective algorithm *max-flow quotient-cut improvement* (`MQI`) [26]. Our algorithm `SOCIAL`, which stands for *faSter mOtif Clustering vIa mAximum fLows*, starts by building a hypergraph model which is an exact representation for the motif-distribution around the seed node on the original graph [7]. Using this model, we create a flow model in which certain cuts correspond one-to-one with sub-sets of the initial cluster that include the seed node and have lower motif conductance than that of the whole cluster. We then use a push-relabel algorithm to either find such a cut and repeat the process recursively, or to prove that the current cluster is optimal among all its sub-clusters containing the seed node. In our experiments with the triangle motif, `SOCIAL` produces communities with a motif conductance value that is 1.7% lower than the state-of-the-art on average, while also being up to multiple orders of magnitude faster.

## 2 Preliminaries

**Graphs.** Let $G = (V = \{0, \ldots, n-1\}, E)$ be an *undirected graph* with no multiple or self edges allowed, such that $n = |V|$ and $m = |E|$. Let $c : V \to \mathbb{R}_{\geq 0}$ be a node-weight function, and let $\omega : E \to \mathbb{R}_{>0}$ be an edge-weight function. We generalize $c$ and $\omega$ functions to sets, such that $c(V') = \sum_{v \in V'} c(v)$ and $\omega(E') = \sum_{e \in E'} \omega(e)$. Let $N(v) = \{u : \{v, u\} \in E\}$ be the *open neighborhood* of $v$, and let $N[v] = N(v) \cup \{v\}$ be the *closed neighborhood* of $v$. We generalize the notations $N(.)$ and $N[.]$ to sets, such that $N(V') = \cup_{v \in V'} N(v)$ and $N[V'] = \cup_{v \in V'} N[v]$. A graph $G' = (V', E')$ is said to be a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. When $E' = E \cap (V' \times V')$, $G'$ is the subgraph *induced* in $G$ by $V'$. Let $\overline{V'} = V \setminus V'$ be the *complement* of a set $V' \subseteq V$ of nodes. Let a *motif* $\mu$ be a connected graph. *Enumerating* the motifs $\mu$ in a graph $G$ consists of building the collection $M$ of all occurrences of $\mu$ as a subgraph of $G$. Let $d(v)$ be the *degree* of node $v$ and $\Delta$ be the maximum degree of $G$. Let $d_\omega(v)$ be the *weighted degree* of a node $v$ and $\Delta_\omega$ be the maximum weighted degree of $G$. Let $d_\mu(v)$ be the *motif degree* of a node $v$, i.e., the number of motifs $\mu \in M$ which contain $v$. We generalize the notations $d(.)$, $d_\omega(.)$, and $d_\mu(.)$ to sets, such that the *volume* of $V'$ is $d(V') = \sum_{v \in V'} d(v)$, the *weighted volume* of $V'$ is $d_\omega(V') = \sum_{v \in V'} d_\omega(v)$, and the *motif volume* of $V'$ is $d_\mu(V') = \sum_{v \in V'} d_\mu(v)$. Let a *spanning forest* of $G$ be an acyclic subgraph of $G$ containing all its nodes. Let the *arboricity* of $G$ be the minimum amount of spanning forests of $G$ necessary to cover all its edges.

**Local Motif Clustering.** In the *local graph clustering* problem, a graph $G = (V, E)$ and a seed node $u \in V$ are taken as input and the goal is to detect a *well-characterized cluster* (or *community*) $C \subset V$ containing $u$. A high-quality cluster $C$ usually contains nodes that are densely connected to one another and sparsely connected to $\overline{C}$. There are many functions to quantify the quality of a cluster, such as *modularity* [5] and *conductance* [22].

The conductance metric is defined as $\phi(C) = |E'|/\min(d(C), d(\overline{C}))$, where $E' = E \cap (C \times \overline{C})$ is the set of edges shared by a cluster $C$ and its complement. *Local motif graph clustering* is a generalization of local graph clustering where a motif $\mu$ is taken as an additional input and the computed cluster optimizes a clustering metric based on $\mu$. In particular, the *motif conductance* $\phi_\mu(C)$ of a cluster $C$ is defined by Benson et al. [4] as a generalization of the conductance in the following way: $\phi_\mu(C) = |M'|/min(d_\mu(C), d_\mu(\overline{C}))$, where $M'$ are all the motifs $\mu$ which contain at least one node in $C$ and one node in $\overline{C}$. Note that, if the motif under consideration is simply an edge, then $|M'|$ is the edge-cut and $\phi_\mu(C) = \phi(C)$.

**Hypergraphs.** Let $H = (\mathcal{V} = \{0, \ldots, \mathfrak{n}-1\}, \mathcal{E})$ be an *undirected hypergraph* with no multiple or self hyperedges allowed, with $\mathfrak{n} = |\mathcal{V}|$ *nodes* and $\mathfrak{m} = |\mathcal{E}|$ *hyperedges* (or *nets*). A net is defined as a subset of $\mathcal{V}$. The nodes that compose a net are called *pins*. Let $c : \mathcal{V} \to \mathbb{R}_{\geq 0}$ be a node-weight function, and let $\omega : \mathcal{E} \to \mathbb{R}_{>0}$ be a net-weight function. We generalize $c$ and $\omega$ functions to sets, such that $c(\mathcal{V}') = \sum_{v \in \mathcal{V}'} c(v)$ and $\omega(\mathcal{E}') = \sum_{e \in \mathcal{E}'} \omega(e)$. A node $v \in \mathcal{V}$ is *incident* to a net $e \in \mathcal{E}$ if $v \in e$. Let $\mathcal{I}(v)$ be the set of incident nets of $v$, let $d(v) := |\mathcal{I}(v)|$ be the *degree* of $v$, and let $d_\omega(v) := \omega(\mathcal{I}(v))$ be the *weighted degree* of $v$. We generalize the notations $d(.)$ and $d_\omega(.)$ to sets, such that the *volume* of $\mathcal{V}'$ is $d(\mathcal{V}') = \sum_{v \in \mathcal{V}'} d(v)$ and the *weighted volume* of $\mathcal{V}'$ is $d_\omega(\mathcal{V}') = \sum_{v \in \mathcal{V}'} d_\omega(v)$. Two nodes are *adjacent* if they are incident to the same net. Let the number of pins $|e|$ in a net $e$ be the *size* of $e$. We define the *contraction* operator as $/$ such that $H/\mathcal{V}'$, with $\mathcal{V}' \subseteq \mathcal{V}$, is the hypergraph obtained by contracting the nodes from $\mathcal{V}'$ of $H$. This contraction consists of substituting all the nodes in $\mathcal{V}'$ by a single representative node $x$, removing nets totally contained in $\mathcal{V}'$, and substituting all the pins in $\mathcal{V}'$ by a single pin $x$ in each of the remaining nets. Given a cluster $\mathcal{V}' \subseteq \mathcal{V}$, the *cut* or *cut-net* $cut(\mathcal{V}')$ of $\mathcal{V}'$ consists of the total weight of the nets crossing the cluster, i.e., $cut(\mathcal{V}') = \sum_{e \in \mathcal{E}'} \omega(\mathcal{E}')$, in which $\mathcal{E}' := \left\{e \in \mathcal{E} : e \cap \mathcal{V}' \neq \emptyset, e \cap \overline{\mathcal{V}'} \neq \emptyset\right\}$.

**Flows.** Let $\mathcal{N} = (V, E)$ be a directed flow network. A directed flow network has one source node $s \in V$, one sink node $t \in V$, and a set of remaining nodes $V \setminus \{s, t\}$. All edges $e = (u, v)$ in a directed flow network are directed and associated with a nonnegative capacity $cap(u, v)$. An s-t flow is a function $f : V \times V \to \mathbb{R}_{>0}$ which satisfies a *capacity* constraint, i. e., $f(u, v) \leq cap(u, v)$, a *symmetry* constraint, i. e., $\forall u, v \in V : f(u, v) = -f(v, u)$, and a *flow conservation* constraint, i. e., $\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(u, v) = 0$. An edge $(u, v)$ is called *saturated* if $cap(u, v) = f(u, v)$; The total amount of flow moved from $s$ to $t$ is defined as the *value* $|f|$ of $f$ and is computed as follows: $|f| = \sum_{u \in V} f(u, t) = \sum_{v \in V} f(s, v)$. A given s-t flow $f$ in $\mathcal{N}$ is *maximum* if, for any s-t flow $f'$ in $\mathcal{N}$, $|f'| \leq |f|$. Let $\mathcal{N}_f = (V, E_f)$ be the *residual graph* associated with a given flow $f$ on $\mathcal{N}$, such that $E_f = \{(u, v) \in V \times V : cap(u, v) - f(u, v) > 0\}$. According to the Max-Flow Min-Cut Theorem [13], the value $|f|$ of a maximum s-t flow $f$ on $\mathcal{N}$ equals the weight of a minimum s-t cut on $\mathcal{N}$, i. e., a 2-way partition of $\mathcal{N}$ where edge weights equal edge capacities, $s$ and $t$ are in distinct blocks, and the total weight of the cut edges is minimum. To find the sink side of the minimum cut associated with a maximum flow in $\mathcal{N}$, a reverse breadth-first search can be performed on $\mathcal{N}$ starting at the sink node $t$.

**Push-Relabel.** For each node $u$ in a directed flow network $\mathcal{N}$, let $d(u)$ be its potential and $exc(u) = \sum_{v \in V}(f(v, u) - f(u, v))$ be its *excess*. A node $u$ is called *active* if $exc(u) > 0$. An edge $(u, v)$ is called *admissible* if $cap(u, v) - f(u, v) > 0$ and $d(u) = d(v) + 1$. The push-relabel [16] algorithm builds a maximum flow by computing a succession of *preflows*, i. e., flows where the flow conservation constraint is relaxed and replaced by $\forall u \in V \setminus \{s, t\} : exc(u) \geq 0$. In the initial preflow, all out-edges of $s$ are saturated, $\forall u \in V \setminus \{s\} : d(u) = 0$, and $d(s) = |V|$.

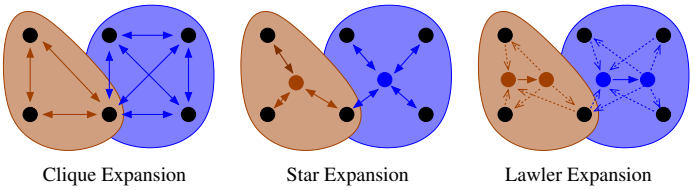



**Figure 1** Net expansion techniques. Nodes and nets of the hypergraph are respectively represented by black circles and colored areas around them. Artificial nodes and edges are respectively represented by circles and arrows with same color as the corresponding net. Bidirectional arrows represent edges in both directions. Solid and dashed edges have finite and infinite weight, respectively.

The initial preflow is evolved via operations *push*, i. e., sending as much flow as possible from an active node through an admissible edge, and *relabel*, i. e., increasing the potential of a node until it becomes active. Preflows induce minimum sink-side cuts, so a maximum flow and a minimum cut are obtained once no node is active.

**Flows on Hypergraphs.** A common technique to solve flow problems on hypergraphs consists of transforming them in directed graphs and then applying traditional graph-based techniques on them. Among the existing transformations [46, 27], we highlight *clique expansion*, *star expansion*, and *Lawler expansion*. In the *clique expansion*, each net is represented by a clique, i.e., a set of edges connecting each pair of its pins in both directions. In this approach, the weight of each edge is equal to weight of the corresponding net $e$ divided by $|e| - 1$ and parallel edges are substituted by a single edge whose weight is the sum of the weights of the removed edges. In the *star expansion*, each net is represented by an auxiliary artificial node connected to its pins by edges in both directions. In this expansion, the edges have the same weight as the corresponding net. In the *Lawler expansion*, each net $e$ is represented by two auxiliary artificial nodes $w_1$ and $w_1$ and a collection of edges. In particular, there is a directed edge $(w_1, w_2)$ which has the same weight as the corresponding net. Additionally, each pin of the corresponding net has an out-edge to $w_1$ and an in-edge from $w_2$, each of them with weight infinity. The three transformation approaches are exemplified in Figure 1.

## 2.1 Related Work

Motif-based clustering has been widely studied in the literature, with works such as [3, 49, 25, 36, 44] partitioning all the nodes of a graph into clusters based on motifs. We also address the topic of clustering based on motifs, but our focus is on identifying clusters in the immediate vicinity of a specific seed node, rather than on the entire graph. Several works [24, 30, 32, 11, 42] propose local clustering algorithms on graphs, but they do not focus on optimizing for motif-based metrics like our work. Instead, they use metrics based on edges, like conductance and modularity. Multiple works [45, 14, 20, 31] propose local clustering algorithms on hypergraphs. These algorithms are not designed for local graph clustering based on motifs, but for local hypergraph clustering. In one of them [45], the authors utilize a hypergraph extension of `FlowImprove` [2], which is itself an extension of the `MQI` [26] technique. Similarly, we also extend `MQI` to hypergraphs, then we use it as one of the steps of our algorithm `SOCIAL`. In this section, we review previous work on local graph clustering based on motifs, which is the focus of our work.

Rohe and Qin [38] propose a local clustering algorithm based on triangle motifs. Their algorithm starts by initializing a cluster containing only the seed node, and iteratively grows this cluster. Particularly, the algorithm greedily inserts nodes contained in at least a

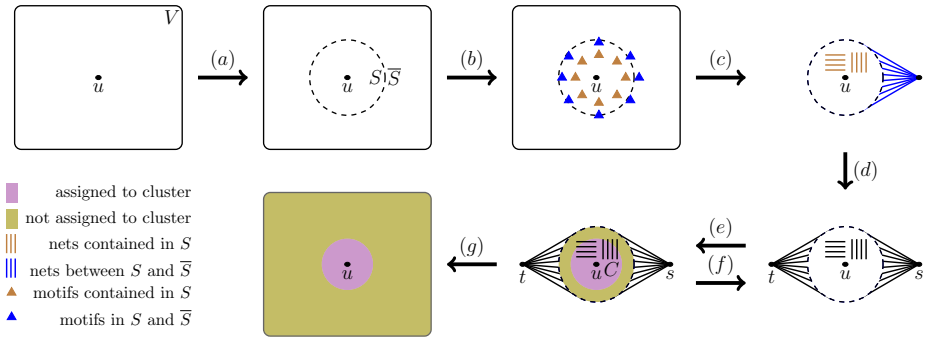



**Figure 2** Illustration of the phases of SOCIAL. (a) Given a seed node $u$ and a graph $G$, a ball $S$ around $u$ is selected. (b) Motif occurrences of $\mu$ with at least a node in $S$ are enumerated. (c) The hypergraph model $H_\mu$ is built by converting motifs into nets and contracting $\overline{S}$ into a single node. The ball $S$ is taken as the initial cluster $C_0$. (d) The flow model $\mathcal{N}$ is built based on $C_0$ in $H_\mu$. (e) A cluster $C \subseteq C_0$ containing $u$ is found using maximum flows. (f) While $C \subset C_0$, the model $\mathcal{N}$ is rebuilt based on $C$, which is taken as the initial cluster $C_0$. (g) When eventually $C = S$, $C$ is converted in a local cluster around the seed node in $G$.

predefined amount of cut triangles. Huang et al. [19] recover local communities containing a seed node in online and dynamic setups based on higher-order graph structures named Trusses [10]. They define the $k$-truss of a graph as its largest subgraph whose edges are all contained in at least $(k-2)$ triangle motifs, hence trusses are a graph structure based on the frequency of triangles. The authors use indexes to search for $k$-truss communities in time proportional to the size of the recovered community.

Yin et al. [49] propose MAPPR, a local motif clustering algorithm based on the Approximate Personalized PageRank (APPR) method. In a preprocessing phase, MAPPR enumerates the motif of interest in the entire input graph and constructs a weighted graph $W$, in which edges only exist between nodes that appear in at least one instance of the motif, and their edge weight is equal to the number of occurrences of the motif containing these two endpoints. Afterward, MAPPR uses an adapted version of the APPR method to find local communities in the weighted graph constructed in the preprocessing phase. MAPPR is able to extract local communities from directed input graphs, something that cannot be done using APPR alone.

Zhang et al. [50] propose LCD-Motif, an algorithm that addresses the local motif clustering problem using a modified version of the spectral method. LCD-Motif has two main differences in comparison to the traditional spectral motif clustering method. First, instead of computing singular vectors, the algorithm performs random walks to identify potential members of the searched cluster. They use the span of a few dimensions of vectors, obtained through random walks, as an approximation for the local motif spectra. Second, instead of using $k$-means for clustering, LCD-Motif searches for the minimum 0-norm vector within the previously mentioned span, which must contain the seed nodes in its support vector.

Meng et al. [33] propose FuzLhocd, a local motif clustering algorithm that uses fuzzy arithmetic to optimize a modified version of modularity. Given a seed node, FuzLhocd starts by detecting probable core nodes of the targeted local community using fuzzy membership. After identifying the probable core nodes of the target local community using fuzzy membership, the algorithm expands these nodes using another fuzzy membership to form a cluster.

Zhou et al. [51] propose HOSPLOC, a local motif clustering algorithm that uses a motif-based random walk to compute a distribution vector, which is then truncated and used in a vector-based partitioning method. The algorithm begins by approximately estimating the

distribution vector through a motif-based random walk. To further refine the computation and focus on the local region, HOSPLOC sets all small vector entries to 0. After this preprocessing step, the algorithm applies a vector-based partitioning method [43] on the resulting distribution vector in order to identify a local cluster.

Shang et al. [41] propose HSEI, a local motif clustering algorithm that uses motif and edge information to grow a cluster from a seed node. The algorithm begins by creating an initial cluster consisting of only the seed node. It then adds nodes to the cluster from the seed's neighborhood, selecting them based on their motif degree. The cluster is expanded using a motif-based extension of the modularity function.

Chhabra et al. [7] propose an algorithm to solve the local motif clustering problem using powerful (hyper)graph partitioning tools [39, 40, 17, 18]. Their algorithm first uses a breadth-first search to select a ball containing the seed node and nearby nodes. Next, they enumerate motif occurrences within the ball and build a (hyper)graph model which allows them to compute the motif conductance of any cluster within the ball. They then partition their model into two blocks using a high-quality (hyper)graph partitioning algorithm, and refine the solution for motif conductance.

# 3 Local Motif Clustering via Maximum Flows

We now present our algorithm SOCIAL, then we discuss its algorithmic components.

## 3.1 Overall Strategy

Given a graph $G = (V, E)$, a seed node $u$, and a motif $\mu$, our strategy for local clustering is based on the following phases. First, we select a set $S \subseteq V$ containing $u$ and close-by nodes. From now on, we refer to this set $S$ as a *ball around* $u$. Second, we enumerate the collection $M$ of occurrences of the motif $\mu$ which contain at least one node in $S$. Third, we build a hypergraph model $H_\mu$ in such a way that the motif-conductance of any cluster $C \subseteq S$ in $G$ can be computed directly in $H_\mu$. Fourth, we set $C_0 = S$ as our initial cluster and use it to build our MQI-based [26] flow model $\mathcal{N}$ from the hypergraph model $H_\mu$. Fifth, we use $\mathcal{N}$ to either find a new cluster $C \subset C_0$ containing $u$ with strictly smaller motif conductance than $C_0$ or prove that such cluster does not exist. While $C \subset C_0$ is found, we take it as our new initial cluster, rebuild $\mathcal{N}$, and repeat the previous phase. When eventually no such strict sub-set is found, the best obtained cluster is directly translated back to $G$ as a local cluster around the seed node. Figure 2 provides a comprehensive illustration of the consecutive phases of SOCIAL. Note that there is no guarantee of finding the best overall cluster including $u$ strictly contained in $S$. Instead, we find a succession of clusters with strictly decreasing cardinality and motif conductance until a local optimum is reached. To better explore the vicinity of $u$ in $G$ and overcome the fact we only find clusters inside $S$, we repeat the overall strategy $\alpha$ times with different balls $S$. Our overall algorithm including the mentioned repetitions is outlined in Algorithm 1.

## 3.2 Hypergraph Model

We follow the same procedure as Chhabra et al. [7] to construct the hypergraph model $H_\mu$. To ensure a thorough understanding of our overall algorithm, we provide a summary of the phases involved, i.e., finding a ball around the seed node, enumerating motifs within it, and finally constructing the hypergraph model $H_\mu$.

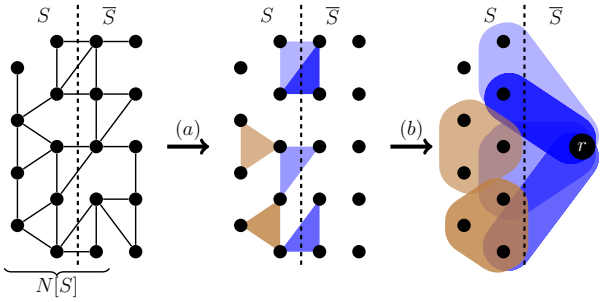



**Figure 3** Example of motif-enumeration and model-construction phases for the triangle motif. In the left, the nodes of $G$ are split into sets $S$ and $\overline{S}$. In the center, motifs containing nodes in $S$ are enumerated. In the right, $H_\mu$ is built by converting motifs in nets and contracting $\overline{S}$ into a node $r$.

**Algorithm 1** Local Motif Clustering via Max Flows.

**Input** graph $G = (V, E)$; seed node $u \in V$; motif $\mu$
**Output** cluster $C^* \subseteq V$

$C^* \leftarrow \emptyset$
**for** $i = 1, \ldots, \alpha$ **do**
  Select ball $S$ around $u$
  $M \leftarrow$ Enumerate motifs in $S$
  Build hypergraph model $H_\mu$ based on $S$ and $M$
  $C \leftarrow S$
  **do**
    $C_0 \leftarrow C$
    Build flow model $\mathcal{N}$ based on $C_0$ in $H_\mu$
    Solve $\mathcal{N}$ to obtain cluster $C \subseteq C_0$ including $u$
  **while** $C \subset C_0$
  **if** $C^* = \emptyset \vee \phi_\mu(C) < \phi_\mu(C^*)$ **then**
    $C^* \leftarrow C$
Convert $C^*$ into a local motif cluster in $G$

**Ball around the Seed Node.** Our approach to select a ball $S$ is a fixed-depth breadth-first search (BFS) rooted on $u$. More specifically, we compute the first $\ell$ layers of the BFS tree rooted on $u$, then we include all its nodes in $S$. For each of the $\alpha$ repetitions of the overall algorithm, we use different amounts $\ell$ of layers for a better algorithm exploration. Two special cases are handled by `SOCIAL`, namely a ball $S$ that is either too small or disconnected from $\overline{S}$. We avoid the first special case by ensuring that $S$ contains 100 or more nodes in at least one repetition of our overall algorithm. More specifically, in case this condition is not automatically met, then we accomplish it in the last repetition by growing additional layers in our partial BFS tree while it contains fewer than 100 nodes. The number 100 is based on the findings of Leskovec et al. [29], which show that most well characterized communities from real-world graphs have a relatively small size, in the order of magnitude of 100 nodes. If the second exceptional case happens, it means that the whole BFS tree rooted on the seed node has at most $\ell$ layers. In this case, we simply stop the algorithm and return the entire ball $S$, which corresponds to an optimal community with motif conductance 0 provided that there is at least one motif in $S$ and another one in $\overline{S}$. The number $\alpha$ of repetitions as well as the amount $\ell$ of layers used in each repetition are tuning parameters.

**Motif Enumeration.** Although enumerating a general motif on some graph is NP-hard [37], there are efficient heuristics to do it such as the one proposed by Kimmig et al. [23]. Nevertheless, simpler motifs such as small paths, cycles, and cliques can be trivially enumerated in

polynomial time. We focus our enumeration phase on the triangle motif. We implement the simple and exact algorithm proposed by Chiba and Nishizeki [8] to enumerate the collection $M$ of occurrences of the motif $\mu$ which contain at least one node in $S$. Roughly speaking, this algorithm works by intersecting the neighborhoods of adjacent nodes. For each node $v$, the algorithm starts by marking its neighbors with degree smaller than or equal to its own degree. For each of these specific neighbors of $v$, it then scans its neighborhood and enumerates new triangles as soon as marked nodes are found. The running time of this algorithm is $O(ma)$, where $a$ is the arboricity of the graph. We apply this enumeration algorithm only on the subgraph induced in $G$ by $N[S]$, which is enough to find all triangles containing at least one node in $S$, as exemplified by transformation (a) in Figure 3. Assuming a constant-bounded arboricity, the overall cost of our motif-enumeration phase for triangles is $O\big(|N[S] \times N[S]) \cap E|\big)$.

**Hypergraph Model.** The hypergraph model $H_\mu$ is finally built in two conceptual operations. First, define a hypergraph containing $V$ as nodes and a set $\mathcal{E}$ of nets such that, for each motif in $M$, $\mathcal{E}$ has a net with pins equal to the endpoints of this motif. Then, we contract together all nodes in $\overline{S}$ into a single node $r$ and substitute parallel nets by a single net whose weight is equal to the summed weights of the removed parallel nets. More formally, we define the hypergraph version of our model as $H_\mu = (S \cup \{r\}, \mathcal{E})$ where the set $\mathcal{E}$ of nets contains one net $e$ associated with each motif occurrence $G' = (V', E') \in M$ such that $e = V'$ if $V' \subseteq S$, and $e = V' \cap S \cup \{r\}$ otherwise. In the former case the net has weight 1, in the latter case the net has weight equal to the amount of motif occurrences in $M$ represented by it. Since node weights in $H_\mu$ are irrelevant for `SOCIAL`, the involved theorems, and the motif conductance metric, we make all node weights unitary in $H_\mu$. In practice, the model $H_\mu$ can be built by instantiating the nodes in $S \cup \{r\}$ and the nets in $\mathcal{E}$. Assuming that the number of nodes in $\mu$ is a constant, our model is built in time $O(|S| + |M|)$ and uses memory $O(|S| + |M|)$. The construction of $H_\mu$ is illustrated in transformation (c) of Figure 2 and demonstrated for a particular example in transformation (b) of Figure 3. Theorem 1 shows that the motif conductance in $G$ of any cluster $C \subseteq S$ can be directly computed from $H_\mu$ assuming $d_\mu(S) \leq d_\mu(\overline{S})$. The assumption $d_\mu(S) \leq d_\mu(\overline{S})$ is fair in practice since the ball $S$ computed via BFS tends to be considerably smaller than $\overline{S}$ for huge sparse networks. Enumerating the motifs in $\overline{S}$ is not reasonable for a local clustering algorithm, but we did verify that our assumption holds during all our experiments.

▶ **Theorem 1** (Theorem 3.2 from [7]). *The motif conductance $\phi_\mu(C)$ of a cluster $C \subseteq S$ in the original graph $G$ can be calculated directly in the hypergraph model $H_\mu$ using the ratio of its cut-net $cut(C)$ to its weighted volume $d_w(C)$, assuming that the motif enumeration step is exact and $d_\mu(S) \leq d_\mu(\overline{S})$.*

## 3.3 Flow Model

In this section, we describe the process of constructing our `MQI`-based flow model $\mathcal{N}$ using the hypergraph model $H_\mu$ and an initial cluster $C_0 \subseteq S$ which contains the seed node $u$. There are three possible implementations of $\mathcal{N}$ based on the three already explained techniques to represent hypergraphs using graphs, namely *clique expansion*, *star expansion*, and *Lawler expansion* (see Figure 1). We show a bijective correspondence between certain s-t cuts in $\mathcal{N}$ and clusters $C \subseteq C_0$ in $G$ that include the seed node $u$ and have motif conductance less than that of $C_0$. We start by converting our hypergraph model $H_\mu$ in a directed graph using the chosen net expansion technique. Second, we find a corresponding cluster $C'_0$

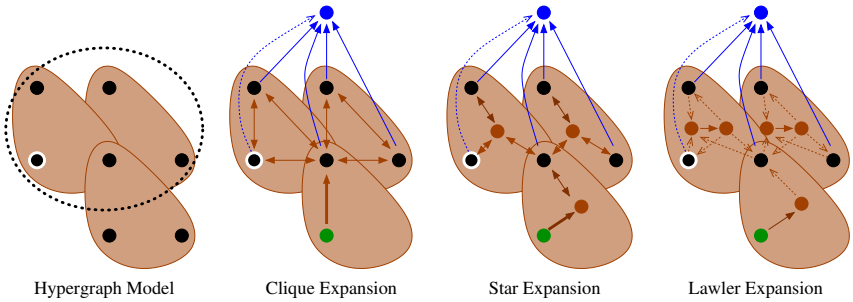



**Figure 4** Flow model $\mathcal{N}$ given a hypergraph model $H_\mu$ and an initial cluster $C_0$. Nodes and nets of $H_\mu$ are respectively represented by black circles and brown areas around them. The seed node $u$ is circled in white and the initial cluster $C_0$ is surrounded by a dotted ellipse. Auxiliary artificial nodes and edges used in each net-expansion are respectively represented by brown circles and arrows. Bidirectional arrows represent pairs of edges in both directions. The seed node $s$, the sink node $t$, and the in-edges of $t$ are respectively represented by a green circle, a blue circle, and blue arrows. Solid and dashed arrows respectively represent edges with finite and infinite weight.

for $C_0$ in the created graph. For the *clique expansion*, $C'_0 = C_0$ since this transformation does not create artificial nodes. For the *star expansion*, $C'_0$ consists of $C_0$ and also the auxiliary artificial nodes connected to at least one node in $C_0$. For the *Lawler expansion*, $C'_0$ consists of $C_0$, the auxiliary artificial nodes $w_1$ having in-edges only from nodes in $C_0$, and the auxiliary artificial nodes $w_2$ having out-edges to at least one node in $C_0$. Third, we contract $\overline{C'_0}$ to a single *source* node $s$ and then remove all its in-edges. Fourth, we multiply the weight of all the remaining edges by $d_{\mathcal{W}}(C_0)$, i.e., the weighted volume of $C_0$ in $H_\mu$. Fifth, we introduce a sink node $t$ and include in-edges to it from each of the nodes $v \in C_0 \setminus \{u\}$, such that the weight of $(v,t)$ is set to $cut(C_0)d_{\mathcal{W}}(v)$, i.e., the cut-net of $C_0$ in $H_\mu$ multiplied by the weighted degree of $v$ in $H_\mu$. Finally, we include an edge $(u,t)$ from the seed node to the sink and set its weight to infinity. Our flow network model $\mathcal{N}$ is concluded by setting edge capacities to match edge weights. Figure 4 shows the three possible configurations of our flow model $\mathcal{N}$ for a given hypergraph model $H_\mu$ and an initial cluster $C_0$.

We now analyze the theoretical guarantees provided by the defined flow model $\mathcal{N}$. Theorem 2 shows that there is a set $C \subset C_0$ in $G$ including the seed node $u$ with motif conductance smaller than that of $C_0$ if, and only if, the value of the maximum flow on $\mathcal{N}$ is less than $cut(C_0)d_{\mathcal{W}}(C_0)$, which is the weight of the trivial cut $(\{s\}, V(\mathcal{N}) \setminus \{s\})$. Due to space constraints, we put the proof of Theorem 2 in our public technical report [6]. In the proof, we show that such improved cluster $C$ consists of the sink side of the minimum cut associated with the maximum flow. This cluster can be directly obtained with a reverse breadth-first search on $\mathcal{N}$ starting at the sink node. For an even stronger claim, see our public technical report [6]. Assumptions (a) and (b) in Theorem 2 are the same used in Theorem 1, which were previously shown to be reasonable in practice. Note that the claim is only valid for motifs with three nodes for clique and star expansion models, while it is valid in general for the Lawler expansion model.

▶ **Theorem 2.** *There is a set $C \subset C_0$ in $G$ including the seed node $u$ with motif conductance smaller than that of $C_0$ if, and only if, the maximum flow on $\mathcal{N}$ is less than $cut(C_0)d_{\mathcal{W}}(C_0)$ under the following assumptions:*

**a)** *the motif enumeration phase is exact;*

**b)** *$d_\mu(S) \leq d_\mu(\overline{S})$ in $G$;*

**c)** *in case $\mathcal{N}$ is based on clique expansion or star expansion, the motif $\mu$ has three nodes;*

**Table 1** Graphs for experiments.

| Graph | $n$ | $m$ | # Triangles |
|---|---|---|---|
| com-amazon | 334 863 | 925 872 | 667 129 |
| com-dblp | 317 080 | 1 049 866 | 2 224 385 |
| com-youtube | 1 134 890 | 2 987 624 | 3 056 386 |
| com-livejournal | 3 997 962 | 34 681 189 | 177 820 130 |
| com-orkut | 3 072 441 | 117 185 083 | 627 584 181 |
| com-friendster | 65 608 366 | 1 806 067 135 | 4 173 724 142 |

SOCIAL utilizes a push-relabel approach to iteratively search for a maximum s-t flow in the model $\mathcal{N}$. If the found maximum flow is strictly smaller than $cut(C_0)\mathrm{d}_{\mathfrak{W}}(C_0)$, then we can directly find a minimum cut with the same weight as it and, consequently, a cluster $C \subset C_0$ containing the seed node $u$ that has a strictly smaller motif conductance value $\phi_\mu(C)$ than that of $C_0$ in $G$. If such a cut is found, the algorithm repeats the process recursively setting the identified sub-cluster $C$ as the new initial cluster, i.e., it constructs a new flow model based on $H_\mu$ and the initial cluster and uses the push-relabel algorithm to continue searching for sub-clusters with even lower motif conductance values. If, on the other hand, the maximum flow is not strictly smaller than $cut(C_0)\mathrm{d}_{\mathfrak{W}}(C_0)$, it means that the current cluster $C_0$ is optimal among all of its sub-clusters containing the seed node $u$, and the algorithm terminates for the given ball $S$.

## 4 Experimental Evaluation

**Methodology.** We implemented SOCIAL in C++. We compiled our program using gcc 11.2 with full optimization turned on (-O3 flag). All our experiments are based on the triangle motif, i.e., the undirected clique of size three. Since this motif has three nodes, Theorem 2 is valid for all net expansion techniques. Therefore, we focus our experiments on the clique expansion technique, which is more efficient than the other techniques because it does not utilize any auxiliary artificial nodes and uses the minimum amount of auxiliary artificial edges. We use the following parameters for SOCIAL: $\alpha = 3$, $\ell \in \{1, 2, 3\}$. We do not utilize more than 3 BFS layers because otherwise the ball around the seed node could become too large in densely connected areas of a graph. Nevertheless, as explained in Section 3.2, SOCIAL might occasionally use more than 3 BFS layers to ensure that the ball includes a minimum of 100 nodes, if the seed node under consideration is located in a sparse portion of the graph. We ensure the integrity of our results by using the same motif-conductance evaluator function for all tested algorithms. In our experiments, we have used a machine with a sixty-four-core AMD EPYC 7702P processor running at 2.0 GHz, 1 TB of main memory, 32 MB of L2-Cache, and 256 MB of L3-Cache. We measure running time, motif-conductance, and/or size of the computed cluster. For each graph, we pick 50 random seed nodes and use all of them as input for each algorithm. When averaging running time or cluster size over multiple instances, we use the geometric mean in order to give every instance the same influence on the *final score*. When averaging motif conductance over multiple instances, the final score is computed via arithmetic mean. This is a necessary averaging strategy since motif conductance can be zero, which makes the geometric mean infeasible to compute. We also use *performance profiles* which relate the running time (resp. motif conductance) of a group of algorithms to the fastest (resp. best) one on a per-instance basis. Their x-axis shows a factor $\tau$ while their y-axis shows the percentage of instances for which algorithm $A$ has up to $\tau$ times the running time (resp. motif conductance) of the fastest (resp. best) algorithm.

**Table 2** Average comparison against state-of-the-art. Times are given in seconds.

| Graph | SOCIAL | | | LMCHGP | | | MAPPR | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\phi_\mu$ | $\|C\|$ | t(s) | $\phi_\mu$ | $\|C\|$ | t(s) | $\phi_\mu$ | $\|C\|$ | t(s) |
| com-amazon | **0.031** | 76 | <0.01 | 0.037 | 64 | 0.22 | 0.153 | 58 | 2.68 |
| com-dblp | **0.090** | 58 | 0.02 | 0.115 | 56 | 0.38 | 0.289 | 35 | 3.04 |
| com-youtube | **0.125** | 1832 | 4.52 | 0.172 | 1443 | 7.93 | 0.910 | 2 | 10.44 |
| com-livejournal | **0.158** | 494 | 3.33 | 0.244 | 387 | 8.17 | 0.507 | 61 | 173.80 |
| com-orkut | 0.273 | 1041 | 256.21 | **0.150** | 13168 | 496.94 | 0.407 | 511 | 923.26 |
| com-friendster | 0.388 | 2060 | 1194.50 | **0.368** | 10610 | 1339.99 | 0.741 | 121 | 16565.99 |
| Overall | **0.178** | 453 | 2.33 | 0.181 | 823 | 12.67 | 0.500 | 50 | 79.34 |

**Instances.** The graphs used in our experiments are the same ones used by Yin et al. [49] and Chhabra et al. [7] and the seed nodes used in our experiments are the same used in [7]. Specifically, we use real graphs from the SNAP Network Dataset Collection [28]. Prior to our experiments, we removed parallel edges, self-loops, and directions of edges and assigning unitary weight to all nodes and edges. Basic properties of the graphs under consideration can be found in Table 1.

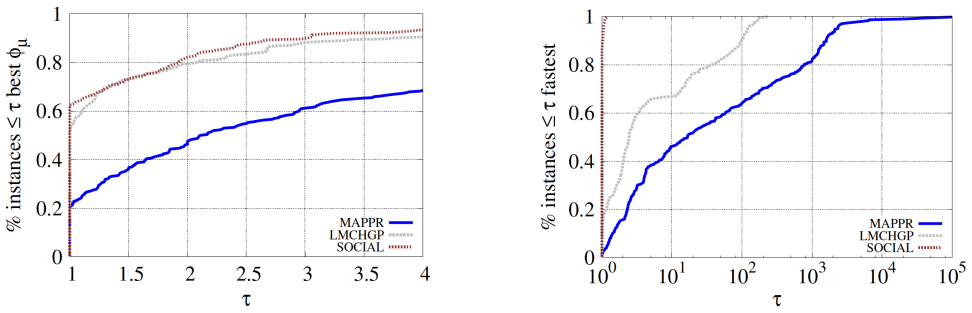
**Competitors.** We experimentally compare our SOCIAL against the state-of-the-art competitors, namely MAPPR [49] and the algorithm proposed by Chhabra et al. [7]. For conciseness, we refer to the latter one from now on as LMCHGP, an acronym for *local motif clustering via (hyper)graph partitioning*. We also ran preliminary experiments with HOSPLOC [51]. However, the algorithm is very slow even for small graphs and not scalable as their algorithm works using an adjacency matrix and hence needs $\Omega(n^2)$ space and time. Moverover, experiments done in their paper are on graphs that are multiple orders of magnitude smaller than the graphs used in our evaluation. Hence, we are not able to run the algorithm on the scale of the instances used in this work. We were not able to explicitly compare against LCD-Motif [50] since their code is not available (neither public, nor privately[1]) and the data presented in the respective paper does not warrant explicit comparisons (e.g. seed nodes are typically not presented in the papers). However, we try to make implicit comparisons in Section 4.1.

We compare our results against the globally best cluster computed for each seed node by MAPPR using its standard parameters ($\alpha = 0.98$, $\epsilon = 10^{-4}$) and by LMCHGP using the configuration with best overall results in [7] (graph model, label propagation, $\alpha = 3$, $\ell \in \{1, 2, 3\}$, and $\beta = 80$). Unless mentioned otherwise, experiments presented here involve all graphs from Table 1.

## 4.1 Results

The performance profile plots shown in Figures 5a and 5b, compare LMCHGP [7] and MAPPR [49] against SOCIAL. In Table 2, we show average results for each graph in our Test Set as well as average results overall. As shown in Figure 5a, SOCIAL obtains the best or equal motif conductance value for 62% of the instances, while LMCHGP and MAPPR respectively obtain the best or equal motif conductance for 49% and 19% of the instances. This result can be explained with two observations. First, SOCIAL explores the solution space considerably better than MAPPR, since we build our model multiple times, while MAPPR simply uses the APPR algorithm. Second, SOCIAL is based on a flow approach which directly optimizes for motif conductance, whereas LMCHGP is based on a (hyper)graph partitioning algorithm which

[1] Personal communication with the authors




**(a)** Motif conductance. **(b)** Running time.

**Figure 5** Performance profiles for motif conductance and running time considering all instances, i.e., 50 random seed nodes for each graph in Table 1. In the running time plot, the `SOCIAL` curve is roughly coincident with the y-axis.

is repeated multiple times to compensate for its design to minimize the number of cut motifs rather than motif conductance. In Table 2, `SOCIAL` outperforms `LMCHGP` for 4 of the 6 graphs and overall, and outperforms `MAPPR` for all graphs and overall. Overall, `SOCIAL` computes clusters with motif conductance 0.178 while `LMCHGP` and `MAPPR` compute clusters with motif conductance 0.181 and 0.500, respectively.

As exhibited in Figure 5b, `SOCIAL` is the fastest one for 87% of the instances, while `LMCHGP` and `MAPPR` are the fastest ones for 12% and 1% of the instances, respectively. Furthermore, the running time of `SOCIAL` is within a factor 1.18 of the running times of the fastest competitors for all instances. `SOCIAL` is respectively up to 237 and 144 063 times faster than `LMCHGP` and `MAPPR` while being a factor 5.4 and 34.1 faster than them on average. The reason for `MAPPR` being considerably slower than the other algorithms is that it must enumerate motifs across the entire graph, while `SOCIAL` and `LMCHGP` only require enumeration of motifs in a ball around the seed node. The reduced but still substantial difference in running time between `SOCIAL` and `LMCHGP` is a result of `LMCHGP`'s repeated partitioning of each ball around the seed node, while `SOCIAL` employs a flow model to greedily improve the motif conductance metric until a local optimum cluster is obtained. In Table 2, `SOCIAL` outperforms `LMCHGP` and `MAPPR` on average in terms of running time for every single graph and overall.

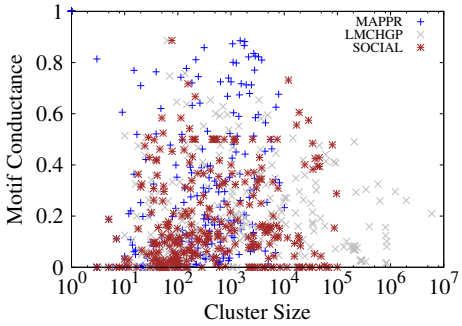



**Figure 6** Motif conductance vs cluster size.

For a more intuitive analysis of the quality of our results, Figure 6 plots motif conductance versus cluster size for all communities computed by the three algorithms. Observe that the communities found by SOCIAL are densely localized in the lower left area of the chart, which is the region with smaller motif conductance and smaller cluster size. On the other hand, communities computed by MAPPR are often in the upper area of the chart and communities computed by LMCHGP are often in the right area of the chart.

**Additional Comparisons.** As we mention above, we were not able to run comparisons against LCD-Motif [50] explicitly since their code is not available (neither publicly, nor privately) and the data presented in the respective papers does not warrant explicit comparisions (e.g., seed nodes are typically not presented in papers, and in this case instances are directed rather than undirected). Here, we make an attempt at implicit comparisons. Zhang et al. [50] (Table 4 therein) compare motif conductance against MAPPR on three directed instances (cit-HepPh, Slashdot, StanfordWeb) and report an geometric mean improvement of 54% in motif conductance for the triangle motif. As SOCIAL works for undirected instances, we have build undirected version of those graphs and run SOCIAL as well as MAPPR for the triangle motif. The geometric mean improvement we obtain over MAPPR is 223% which is significantly larger than the improvement of Zhang et al. [50] over MAPPR. Also note that in our experiments from Table 2, the geometric mean improvement (using the average motif conductance values) of SOCIAL over MAPPR in motif conductance is 219%.

## 5 Conclusion

In this work, we propose SOCIAL, a fast flow-based algorithm to solve the local motif clustering problem in graphs. Given a seed node, our SOCIAL selects a ball of nodes around it, which is taken as an initial cluster and used to build an exact hypergraph model where nets represent motifs. Using this model and the initial cluster, we create a flow model in which the value of the maximum s-t flow is directly related to the presence of sub-sets of the initial cluster that contain the seed node and have lower motif conductance than the initial cluster as a whole. Utilizing a push-relabel algorithm, SOCIAL either identifies a sub-cluster containing the seed node with improved motif conductance and repeats the process recursively by considering it as the initial cluster, or demonstrates that the current initial cluster is the best among all its sub-clusters that include the seed node.

In our experiments with the triangle motif, we found that SOCIAL produces communities with an average motif conductance better than the state-of-the-art, while running up to orders of magnitude faster on average. For future work, we intend to conduct experiments with larger motifs and use the Lawler-expansion version of our flow model, since it is the only one whose quality guarantee holds true for larger motifs. Laslty, we intend to add parallelization to improve the speed on large instances further.

## References

1 Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486, 2006. doi:10.1109/FOCS.2006.44.

2 Reid Andersen and Kevin J. Lang. An algorithm for improving graph partitions. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 651–660. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347154.

3 Austin R. Benson, David F. Gleich, and Jure Leskovec. Tensor spectral clustering for partitioning higher-order network structures. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 118–126. SIAM, 2015. doi:10.1137/1.9781611974010.14.

4 Austin R. Benson, David F. Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016. doi:10.1126/science.aad9029.

5 Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Trans. Knowl. Data Eng.*, 20(2):172–188, 2008. doi:10.1109/TKDE.2007.190689.

6 Adil Chhabra, Marcelo Fonseca Faraj, and Christian Schulz. Faster local motif clustering via maximum flows. *CoRR*, abs/2301.07145, 2023. doi:10.48550/arXiv.2301.07145.

7 Adil Chhabra, Marcelo Fonseca Faraj, and Christian Schulz. Local motif clustering via (hyper)graph partitioning. In *Symposium on Algorithm Engineering and Experiments (ALENEX 23), January 22-23, 2023*. SIAM, 2023.

8 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comp.*, 14(1):210–223, 1985. doi:10.1137/0214017.

9 Fan Chung and Olivia Simpson. Solving linear systems with boundary conditions using heat kernel pagerank. In *Intl. Workshop on Algorithms and Models for the Web-Graph*, pages 203–219. Springer, 2013. doi:10.1007/978-3-319-03536-9_16.

10 Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National security agency Tech. report*, 16(3.1), 2008. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.505.7006&rep=rep1&type=pdf.

11 Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. Local search of communities in large graphs. In *ACM SIGMOD Intl. Conf. on Management of data*, pages 991–1002, 2014. doi:10.1145/2588555.2612179.

12 Alessandro Epasto, Jon Feldman, Silvio Lattanzi, Stefano Leonardi, and Vahab Mirrokni. Reduce and aggregate: similarity ranking in multi-categorical bipartite graphs. In *WWW*, pages 349–360, 2014. doi:10.1145/2566486.2568025.

13 Lester Randolph Ford and Delbert Ray Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

14 Kimon Fountoulakis, Pan Li, and Shenghao Yang. Local hyper-flow diffusion. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27683–27694, 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/e924517087669cf201ea91bd737a4ff4-Abstract.html.

15 Kimon Fountoulakis, Meng Liu, David F. Gleich, and Michael W. Mahoney. Flow-based algorithms for improving clusters: A unifying framework, software, and performance. *SIAM Rev.*, 65(1):59–143, 2023. doi:10.1137/20m1333055.

16 Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.

17 Lars Gottesbüren, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Scalable shared-memory hypergraph partitioning. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX*, pages 16–30. SIAM, 2021. doi:10.1137/1.9781611976472.2.

18 Lars Gottesbüren, Tobias Heuer, Peter Sanders, Christian Schulz, and Daniel Seemaier. Deep multilevel graph partitioning. In *29th Annual European Symposium on Algorithms, ESA*, volume 204 of *LIPIcs*, pages 48:1–48:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.48.

19 Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. Querying k-truss community in large and dynamic graphs. In *ACM SIGMOD*, pages 1311–1322, 2014. doi:10.1145/2588555.2610495.

20 Rania Ibrahim and David F. Gleich. Local hypergraph clustering using capacity releasing diffusion. *CoRR*, abs/2003.04213, 2020. arXiv:2003.04213.

21 Lucas G. S. Jeub, Prakash Balachandran, Mason A. Porter, Peter J. Mucha, and Michael W. Mahoney. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Physical Review E*, 91(1):012821, 2015. doi:10.1103/PhysRevE.91.012821.

22 Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *JACM*, 51(3):497–515, 2004. doi:10.1145/990308.990313.

23 Raphael Kimmig, Henning Meyerhenke, and Darren Strash. Shared memory parallel subgraph enumeration. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops*, pages 519–529. IEEE Computer Society, 2017. doi:10.1109/IPDPSW.2017.133.

24 Kyle Kloster and David F. Gleich. Heat kernel based community detection. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 1386–1395. ACM, 2014. doi:10.1145/2623330.2623706.

25 Christine Klymko, David F. Gleich, and Tamara G. Kolda. Using triangles to improve community detection in directed networks. *CoRR*, abs/1404.5874, 2014. arXiv:1404.5874.

26 Kevin J. Lang and Satish Rao. A flow-based method for improving the expansion or conductance of graph cuts. In *Integer Programming and Combinatorial Optimization, 10th International IPCO*, volume 3064 of *LNCS*, pages 325–337. Springer, 2004. doi:10.1007/978-3-540-25960-2_25.

27 Eugene L. Lawler. Cutsets and partitions of hypergraphs. *Networks*, 3(3):275–285, 1973. doi:10.1002/net.3230030306.

28 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

29 Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.*, 6(1):29–123, 2009. doi:10.1080/15427951.2009.10129177.

30 Yixuan Li, Kun He, David Bindel, and John E. Hopcroft. Uncovering the small community structure in large networks: A local spectral approach. In *Proceedings of the 24th International Conference on World Wide Web, WWW*, pages 658–668. ACM, 2015. doi:10.1145/2736277.2741676.

31 Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. Strongly local hypergraph diffusions for clustering and semi-supervised learning. In *WWW '21: The Web Conference*, pages 2092–2103. ACM / IW3C2, 2021. doi:10.1145/3442381.3449887.

32 Michael W. Mahoney, Lorenzo Orecchia, and Nisheeth K. Vishnoi. A local spectral method for graphs: with applications to improving graph partitions and exploring data graphs locally. *J. Mach. Learn. Res.*, 13:2339–2365, 2012. doi:10.5555/2503308.2503318.

33 Tao Meng, Lijun Cai, Tingqin He, Lei Chen, and Ziyun Deng. Local higher-order community detection based on fuzzy membership functions. *IEEE Access*, 7:128510–128525, 2019. doi:10.1109/ACCESS.2019.2939535.

34 Mrudula Murali, Katerina Potika, and Chris Pollett. Online local communities with motifs. In *2020 Second Intl. Conf. on Transdisciplinary AI (TransAI)*, pages 59–66. IEEE Computer Society, September 2020. doi:10.1109/TransAI49837.2020.00014.

35 Lorenzo Orecchia and Zeyuan Allen Zhu. Flow-based algorithms for local graph clustering. In *SODA*, pages 1267–1286. SIAM, 2014. doi:10.1137/1.9781611973402.94.

36 Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007. doi:10.1093/bioinformatics/btl301.

37 Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *J. Graph Theory*, 1(4):339–363, 1977. doi:10.1002/jgt.3190010410.

38 Karl Rohe and Tai Qin. The blessing of transitivity in sparse and stochastic networks. *arXiv preprint*, 2013. arXiv:1307.2302.

39 Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In *Algorithms – ESA 2011 – 19th Annual European Symposium*, volume 6942 of *LNCS*, pages 469–480. Springer, 2011. doi:10.1007/978-3-642-23719-5_40.

40 Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. $k$-way hypergraph partitioning via $n$-level recursive bisection. In *Proceedings of the Workshop on Algorithm Engineering and Experiments, ALENEX*, pages 53–67. SIAM, 2016. doi:10.1137/1.9781611974317.5.

41 Ronghua Shang, Weitong Zhang, Jingwen Zhang, Jie Feng, and Licheng Jiao. Local community detection based on higher-order structure and edge information. *Physica A: Statistical Mechanics and its Applications*, 587:126513, 2022. doi:10.1016/j.physa.2021.126513.

42 Mauro Sozio and Aristides Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 939–948. ACM, 2010. doi:10.1145/1835804.1835923.

43 Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26, 2013. doi:10.1137/080744888.

44 Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web, WWW*, pages 1451–1460. ACM, 2017. doi:10.1145/3038912.3052653.

45 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1708–1718. ACM, 2020. doi:10.1145/3394486.3403222.

46 Nate Veldt, Austin R. Benson, and Jon M. Kleinberg. Hypergraph cuts with general splitting functions. *SIAM Rev.*, 64(3):650–685, 2022. doi:10.1137/20m1321048.

47 Konstantin Voevodski, Shang-Hua Teng, and Yu Xia. Spectral affinity in protein networks. *BMC systems biology*, 3(1):1–13, 2009. doi:10.1186/1752-0509-3-112.

48 Dorothea Wagner and Frank Wagner. Between min cut and graph bisection. In *Mathematical Foundations of Computer Science 1993, International Symposium, MFCS*, volume 711 of *LNCS*, pages 744–750. Springer, 1993. doi:10.1007/3-540-57182-5_65.

49 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564. ACM, 2017. doi:10.1145/3097983.3098069.

50 Yunlei Zhang, Bin Wu, Yu Liu, and Jinna Lv. Local community detection based on network motifs. *Tsinghua Science and Technology*, 24(6):716–727, 2019. doi:10.26599/TST.2018.9010106.

51 Dawei Zhou, Si Zhang, Mehmet Yigit Yildirim, Scott Alcorn, Hanghang Tong, Hasan Davulcu, and Jingrui He. High-order structure exploration on massive graphs: A local graph clustering perspective. *ACM Trans. Knowl. Discov. Data*, 15(2):18:1–18:26, 2021. doi:10.1145/3425637.

# Primal-Dual Schemes for Online Matching in Bounded Degree Graphs

**Ilan Reuven Cohen**
Bar-Ilan University, Ramat Gan, Israel

**Binghui Peng**
Columbia University, New York, NY, USA

## Abstract

We explore various generalizations of the online matching problem in a bipartite graph $G$ as the b-matching problem [8], the allocation problem [5], and the AdWords problem [13] in a beyond-worst-case setting. Specifically, we assume that $G$ is a $(k, d)$-bounded degree graph, introduced by Naor and Wajc [14]. Such graphs model natural properties on the degrees of advertisers and queries in the allocation and AdWords problems. While previous work only considers the scenario where $k \geq d$, we consider the interesting intermediate regime of $k \leq d$ and prove a tight competitive ratio as a function of $k, d$ (under the small-bid assumption) of $\tau(k, d) = 1 - (1 - k/d) \cdot (1 - 1/d)^{d-k}$ for the $b$-matching and allocation problems. We exploit primal-dual schemes [6, 3] to design and analyze the corresponding tight upper and lower bounds. Finally, we show a separation between the allocation and AdWords problems. We demonstrate that $\tau(k, d)$ competitiveness is impossible for the AdWords problem even in $(k, d)$-bounded degree graphs.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online Matching, Primal-dual analysis, bounded-degree graph, the AdWords problem

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.35

**Funding** *Ilan Reuven Cohen*: Supported by the Israel Science Foundation grant No. 1737/21.
*Binghui Peng*: Supported by NSF IIS-1838154, CCF-2106429, CCF-2107187, CCF-1763970, CCF-2212233.

## 1 Introduction

Ad auctions have emerged as a powerful tool for online advertisers seeking to reach targeted audiences through search engines and other digital platforms. Therefore, there has been much interest in optimizing their revenue generation in recent years. In an ad auction, advertisers bid on specific keywords or phrases relevant to their products or services. When users query search engines, in addition to the algorithmic search results, it also displays sponsored ads that match the ad auctions. These ads are rapidly sold or assigned to potential buyers (advertisers). In their seminal work, Mehta et al. [13] developed a model for optimizing the allocation of ad auctions, building upon the concept of online bipartite matching of Karp et al. [10]. There are $n$ bidders, where each bidder $i \in \{1, \ldots, n\}$, has a known daily budget $B(i)$. Ad auctions arrive online, one at a time, and each bidder $i$ provides a bid $b(i, j)$ for buying the product $j$ (displaying the ad). Once the bids are received, the seller's algorithm must allocate the ad to one of the interested bidders, and this decision is irrevocable. The seller's objective is to maximize the total revenue accumulated from the ad auctions. To accomplish this, Mehta et al. [13] proposed a (tight) deterministic algorithm, which is $(1 - 1/e)$-competitive in cases where the budget of each bidder is relatively larger than the bids. This is a realistic assumption in the ad auction scenario.


Consequently, a loss of $1/e$ can equate to billions of dollars in potential revenue loss each year. In addition, the $1 - 1/e$ hardness result of Karp et al. [10] for online matching is prevalent in all the problem extensions. Therefore, researchers have been inspired to pursue a more comprehensive theoretical understanding that extends beyond the worst-case scenarios.

One line of research considers stochastic arrival models, which include random order arrival and iid arrival models. In the random order model, a fixed input graph is prepared in advance, and the online vertices are randomly arranged. On the other hand, the iid arrival model involves drawing online vertices from a known or unknown distribution. With these arrival models, researchers were able to demonstrate results that outperform the $1 - 1/e$ competitive ratio in both online matching and vertex-weighted matching, as demonstrated by various sources [4, 7, 9, 11, 12].

In this work, we consider a different line of research, which assumes structural properties commonly encountered in ad allocation instances used for targeted advertising. Such properties were first identified and formalized by Naor and Wajc [14]. We assume that $G$ is a bipartite $(k, d)$-bounded-degree graph (for brevity's sake, we use the term $(k, d)$-graphs).

Specifically, we assume that advertisers are interested in a large number of ad slots (i.e. $\sum_j b(i, j) \geq k \cdot B(i)$ for all $i$) and every ad slot is of interest to a relatively small number of advertisers (i.e. $|N(j)| = |\{i : b(i, j) > 0\}| \leq d$ for all $j$).

This structural assumption is relatively natural for ad allocation graphs. First, ad campaigns often target relatively small, specific population segments, resulting in users belonging to fairly few segments. When coupled with the limited number of active campaigns, this creates a limited pool of ads that may be shown to a particular user, justifying the assumption of a small degree for ad slots. Second, advertisers generally aim to target large population segments. However, they often allocate insufficient budget to display ads to all users within a segment. This leads to the high-degree assumption on the offline side since every page view by a specific, targeted user corresponds to a vertex in the graph.

**Related work.** Buchbinder et al. [5] developed a primal-dual algorithm for AdWords that attains a competitive ratio of $(1-1/c)(1-R_{\max})$, where $c = (1+R_{\max})^{1/R_{\max}}$. As $R_{\max} \to 0$, the competitiveness tends to $1 - 1/e$. Buchbinder et al. also examined a setting where the degree of each incoming query was upper bounded by $d$ (i.e., $(1, d)$-graph) and produced an algorithm with a competitive ratio of nearly $1 - (1 - 1/d)^d$ for the AdWords with equal bids per advertiser (a.k.a, the allocation problem). Azar et al. [3] showed that this ratio is the best possible, also for randomized algorithms. The expression $1 - (1 - 1/d)^d$ is always greater than $1 - 1/e$ but approaches the latter as $d$ increases. Naor and Wajc [14] introduced the $(k, d)$-graphs class and applied it to the AdWords problem and online bipartite matching. For $k \geq d$, they developed deterministic online algorithms achieving a competitive ratio of $1 - (1 - 1/d)^k$ for online bipartite matching and its vertex-weighted extension, where all edges connected to an offline vertex $i$ have the same weight. They also showed that this ratio holds for AdWords with equal bids per advertiser, where each advertiser has the same bid for all relevant keywords. The ratio of $1 - (1 - 1/d)^k$ is the best possible for online bipartite matching and the vertex-weighted extension if $k \geq d$. For AdWords with arbitrary bids, Naor and Wajc provided an algorithm with a competitive ratio of $(1 - R_{\max})(1 - (1 - 1/d)^k)$. They also demonstrated that if $k \geq d$, the best possible competitiveness is upper-bounded by $(1 - R_{\max})(1 - (1 - 1/d)^k/R_{\max})$. As $k/d$ increases, the expression $1 - (1 - 1/d)^k$ tends to 1, and for $k \simeq d$ it approaches $1 - 1/e$.

A recent work [2], studied the online $b$-matching problem in $(k, d)$-graphs for $k \geq d$. In this scenario, each vertex $a$ located on the left-hand side of the bipartite graph represents a server with a capacity of $b_a$, which implies that it can be matched with up to $b_a$ incoming

| | $\alpha = 1/3$ | $\alpha = 1/2$ | $\alpha = 2/3$ | $\alpha = 3/4$ |
|---|---|---|---|---|
| $d = 12$ | 0.66765 | 0.70335 | 0.76464 | 0.80744 |
| $d = 24$ | 0.66258 | 0.69997 | 0.76286 | 0.80634 |
| $d = 48$ | 0.69997 | 0.69833 | 0.76200 | 0.80581 |
| $d = 120$ | 0.65868 | 0.69737 | 0.76149 | 0.8055 |
| $d \to \infty$ | 0.65772 | 0.69673 | 0.76116 | 0.8053 |

**Figure 1** Comparison of $\tau(k, d)$ for various $d, k = \alpha \cdot d$ values.

requests that appear as right-hand side vertices. The objective is to maximize the size of the generated matching. Their results also hold for the vertex-weighted problem extension and, thus, for AdWords and auction problems in which each bidder issues individual, equally valued bids. They designed deterministic online algorithms for optimal competitiveness, for $k \geq d$ instances. Later, in [1], they showed a non-trivial extension for the AdWords problem, i.e., bids of arbitrary value. Specifically, they showed an algorithm for the general AdWords problem that achieves competitiveness arbitrarily close to 1 for $k \geq d$ using the small-bids assumption.

**Our Contributions.** We examine the interesting scenarios of $(k, d)$-graphs where $k \leq d$. First, we study the $b$-matching problem for $k \in \mathbb{N}$. Our algorithm is based on a fractional algorithm for $b$-matching. In Section 3, we provide a fractional algorithm with a competitive ratio of $\tau(k, d)$, where $\tau(k, d) = 1 - \frac{d-k}{d}\left(1 - \frac{1}{d}\right)^{d-k}$. The algorithm is based on two steps; The first ensures that each offline vertex's fractional load at the end of the algorithm is at least $k/d$, while the second step uses the water-level algorithm to allocate the remaining volume. We analyze the algorithm using the primal-dual scheme and a carefully chosen potential function, $f^{(k,d)}$. Second, we prove that the bound $\tau(k, d)$ is tight. We use the primal-dual techniques from [3] to analyze a class of $(k, d)$-graphs where any online algorithm is at most $\tau(k, d)$-competitive.

Next, for arbitrary $k \in \mathbb{R}_+$, let $\alpha = k/d \leq 1$, we prove that for AdWords in the equally-valued bids case, it is possible to achieve almost $\tau(\alpha)$-competitive values (under the small bid assumption), where $\tau(\alpha) = 1 - (1 - \alpha) \cdot e^{\alpha - 1}$. It is evident that $\lim_{d \to \infty, k = \alpha \cdot d} \tau(k, d) = \tau(\alpha)$. Figure 1 presents a comparison of the bound of $\tau(k, d)$ as a function of $\alpha = k/d$ and $d$. The algorithm uses a similar approach to our fractional matching algorithm on $(k, d)$-graph and uses a 'smoother' potential function $f^{(\alpha)}$ as a function only of $\alpha$. We round the fractional algorithm outputs using the algorithm of [5].

Finally, we prove that achieving a $\tau(\alpha)$-competitive algorithm for the AdWords problem for bids of arbitrary value is impossible. To do that, we significantly extend the lower bounds of [3], which essentially uses a single scenario, to a multi-scenario instance; we show that the primal-dual techniques can be adapted to analyze the multi-scenario. Subsequently, we prove an improved upper bound by carefully choosing such a multi-scenario instance.

## 2 Preliminaries

### The AdWords problem

The online ad auctions problem comprises a group of $L = \{1, \ldots, n\} = [n]$ buyers, where bidder $i$ has a daily budget of $B(i)$. In the online setting, a group of $R = [m]$ products arrives one by one, and each buyer $i$ places a bid $b(i, j)$ for each product $j$. The objective is to allocate the products to buyers to maximize the seller's revenue. The allocation can be

| Primal: | Dual: |
|---|---|
| $\max \sum_{j \in R} \sum_{i \in N(j)} b(i,j) \cdot x_{i,j}$ | $\min \sum_{i \in L} B(i) \cdot y_i + \sum_{j \in R} z_j$ |
| $\sum_{j \in N(i)} b(i,j) \cdot x_{i,j} \leq B(i), \ \forall i \in L \quad (y_i)$ | $b(i,j) \cdot y_i + z_j \geq b(i,j), \quad \forall i,j \quad (x_{i,j})$ |
| $\sum_{i \in N(j)} x_{i,j} \leq 1, \quad \forall j \in R \quad (z_j)$ | $y_i \geq 0, \quad \forall i \in L$ |
| $x_{i,j} \geq 0, \quad \forall i,j$ | $z_j \geq 0, \quad \forall j \in R$ |

**Figure 2** The fractional AdWords problem (the primal) and the corresponding dual problem.

integral, where a product goes to a single buyer, or fractional, where products can be divided among several buyers. However, the sum of the fractions cannot exceed 1 for each product. The revenue received from a buyer $i$ is the minimum between the sum of the costs of the products allocated to the buyer (times the allocated fraction) and the buyer's total budget. The problem can be formulated as a linear programming problem where the variable $x_{i,j}$ denote the fraction of product $j$ allocated to buyer $i$. The objective is to maximize the total seller revenue, and the constraints guarantee that the sum of fractions for each product does not exceed 1 and each buyer's budget is not exceeded. See Figure 2 for the corresponding fractional Linear Program and its dual.

▶ **Definition 1** ($(k,d)$-bounded graph [14]). *A bipartite graph $G = (L, R, E)$ is $(k,d)$-bounded if each left vertex $i \in L$ has a degree $d(i) \geq k$, and every right vertex $j \in R$ has a degree $d(j) \leq d$. For ad allocations, we replace $d(i) \geq k$ with the property $\sum_j b(i,j) \geq k \cdot B(i)$.*

**Rounding the fractional solution.** For the AdWords problem with equal bids per advertiser, in [5], they presented an algorithm that rounds the online fractional solution, such that if each product's bid price is small compared to the total bidder budget (the small bid assumption), then this rounding phase only reduces the revenue by a factor of $1 - o(1)$ compared to the fractional solution revenue. Formally, let $b_{\max} = \max_j b(j)$ denote the maximum bid price, and $B_{\min} = \min_i B(i)$.

▶ **Theorem** (Theorem 5.4 in [5]). *There exists a deterministic online rounding algorithm, where the integral allocation algorithm's revenue is at least $1 - o(1)$ times the revenue of the fractional solution, provided that:* $(1 + b_{\max})\sqrt{\frac{\ln 2n}{B_{\min}}} = o(1)$

## 3 Matching on bounded-degree graphs

In this section, we prove the tight bounds for online fractional bipartite matching on bounded-degree graphs.

### The b-matching problem

Consider a bipartite graph $G = (L \cup R, E)$, where $L$ represents servers and $R$ represents requests. The set of servers, $L$, is known beforehand, and each server $i \in L$ has an individual capacity $B(i)$, indicating that it can be matched with up to $B(i)$ requests. Requests arrive online, one by one, and when a new request $j \in R$ arrives, its incident edges are revealed, and

it must be immediately and irreversibly matched to an eligible server, provided one exists. The objective is to maximize the number of matching edges. According to the AdWords formulation, we have $b(i,j) = 1$ if $(i,j) \in E$ and 0 otherwise.

A fractional version of the bipartite matching problems allows for every node $j \in R$ to be allocated partially with nodes $i \in N(j)$ in fractions $x_{i,j}$. We study the case where $G$ is a $(k,d)$-graph. Let $\tau(k,d) = 1 - \frac{d-k}{d}\left(1 - \frac{1}{d}\right)^{d-k}$.

▶ **Theorem 2.** *For the fractional online b-matching problem in $(k,d)$-graphs, where $k \leq d$ and $k, d \in \mathbb{N}_+$, there exists a $\tau(k,d)$-competitive algorithm.*

In Section 4, we prove the result is tight for any $(k,d)$ graph. Note that we may apply the rounding algorithm of [5] (Theorem 5.4) to the $b$-matching problem.

## 3.1 The Two-Step-Water-Level algorithm

For a vertex $i \in L$, denote the level of vertex $i$ as $\ell(i) = \frac{\sum_{j \in N(i)} x_{ij}}{B(i)}$. Note that, for $(k,d)$-graphs, a naive online algorithm can ensure that the level of any $i \in L$ at the end of the algorithm is $k/d$. This is achieved by assigning each online vertex a $1/d$ fraction to each neighbor. Since the degree of any $j \in R$ is at most $d$, the algorithm is feasible. In addition, since for any vertex $i \in L$, the degree is at least $k \cdot B(i)$, its level at the end of the algorithm will be $k/d$. Accordingly, our algorithm consists of two steps. For the allocation of an online vertex $j \in R$, the first step fractionally allocates the vertex's volume to vertices in $i \in N(j)$ such that their degree is less or equal to $k \cdot B(i)$. As a result of this step, a level of at least $k/d$ is guaranteed at the end of the algorithm's run for all $i \in L$. The second step runs the water-level algorithm, continuously raising the level on the current set of minimum-level vertices in $N(j)$ on the remainder of the vertex $j$ volume. We use the notations $d^j(i)$ and $\ell^j(i)$ for the degree and level of vertex $i$ after vertex $j$'s arrival, respectively. Given a function $f^{(k,d)} : [0,1] \to [0,1]$, which we will describe later, the algorithm also maintains a solution to the dual variables.

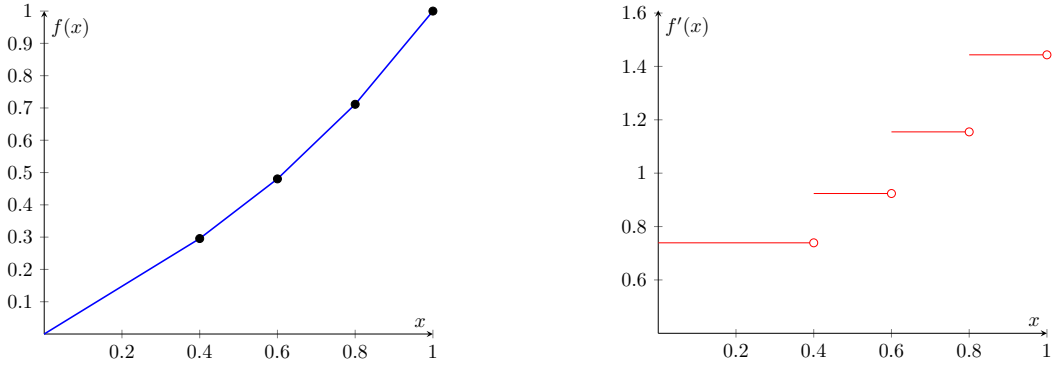
**Algorithm 1** The **Two-Step-Water-Level** algorithm.

Upon the arrival of a new vertex $j \in R$:

1. For each vertex $i \in N(j)$, such that $d^j(i) \leq k \cdot B(i)$
   Increase (if necessary) the level of $i$ to $d^j(i)/d$
2. Using the leftover volume:
   Increase the level of vertices in $N(j)$ using the water-level algorithm
3. Update the dual variables, let $\ell(j) = \min_{i \in N(j)} \ell^j(i)$ and $\hat{\ell}(j) = \max(k/d, \ell(j))$.
   Set $y_i = f^{(k,d)}(\ell(i))$, for $i \in N(j)$
   Set $z_j = 1 - f^{(k,d)}(\hat{\ell}(j))$

## 3.2 The algorithm's feasibility

We will prove that for a proper $f^{(k,d)}$,(for brevity's sake, we use $f$ for $f^{(k,d)}$ for the rest of the section) the primal and dual solutions of the **Two-Step-Water-Level** (TSWL) algorithm are feasible.




**(a)** $f(x)$. **(b)** $f'(x)$.

**Figure 3** The function $f^{(k,d)}(x)$ for $k = 2$ and $d = 5$.

▶ **Lemma 3.** *Given a function $f : [0,1] \to [0,1]$, where $f$ is a monotone non-decreasing function, the primal and dual solutions of the TSWL Algorithm are feasible at the end of the algorithm.*

**Proof.** First, we will show that the algorithm does not over-allocate online vertex $j$ in step 1. We assume that the algorithm is feasible until the arrival of vertex $j$. For any vertex $i \in N(j)$, its degree increases by 1, and by the algorithm definition, its level prior to the assignment of vertex $i$ is at least $\frac{d^j(i)-1}{d}$. Therefore, for any such $i$, $x_{ij}$ after the first step is at most $\frac{d^j(i)}{d} - \frac{d^j(i)-1}{d} = \frac{1}{d}$, and by the definition of $(k,d)$-graphs, $|N(j)| \leq d$. Therefore, vertex $j$ fractional allocation at step 1 is at most $|N(j)| \cdot \frac{1}{d} \leq 1$. By the definition of step 1, and since $G$ is $(k,d)$-graph, at the end of the sequence, the level of any vertex $i \in L$ is at least $k/d$.

To prove that the dual is feasible, we need to verify for $i \in N(j)$ that $y_i + z_j \geq 1$ $(b(i,j) = 1)$. If $\ell(j) \geq k/d$, then

$$y_i + z_j = f(\ell(i)) + 1 - f(\ell(j)) \geq f(\ell^j(i)) + 1 - f(\ell(j)) \geq 1,$$

where the first inequality is since $\ell(i) \geq \ell^j(i)$ and $f$ is a monotone, non-decreasing function, and the second inequality is because, by definition, $\ell^j(i) \geq \ell(j)$. If $\ell(j) < k/d$, by definition $\hat{\ell}(j) = k/d$, then we have

$$y_i + z_j = f(\ell(i)) + 1 - f(\hat{\ell}(j)) \geq f(k/d) + 1 - f(k/d) \geq 1,$$

the first inequality is since the level of any vertex $i \in L$ at least $k/d$ and the end of the algorithm, and since $f$ is a monotone, non-decreasing function. ◀

## 3.3 The potential function

For a function $f$, we denote $f'_+(x)(f'_-(x))$ the right (left) derivative of $f$ at point $x$.

▶ **Lemma 4.** *For any $k \leq d, k \in \mathbb{N}_+$ there exists $f^{(k,d)} = f : [0,1] \to [0,1]$, such that*

- *$f, f'$ are monotone, non-decreasing functions.*
- *$f(0) = 0, f(1) = 1$.*
- *$1 - f\left(\frac{d-i}{d}\right) + f'_-\left(\frac{d-i}{d}\right) = \frac{1}{\tau(k,d)}$, for $i \in \{0, \ldots, d-k\}$.*

**Proof.** Let $\beta = 1/\tau(k,d)$. We define $f[0,1] \to [0,1]$ as a piece-wise linear function.

$$f'_+\left(\frac{d-i+1}{d}\right) = f'_-\left(\frac{d-i}{d}\right) = \beta \cdot \left(\frac{d-1}{d}\right)^i, \quad \text{for } i \in \{0,\dots,d-k-1\},$$

and

$$f'_+(0) = f'_-\left(\frac{k}{d}\right) = \beta \cdot \left(\frac{d-1}{d}\right)^{d-k}.$$

By definition, $f'$ is a monotone, non-decreasing function, and since $f'(x) > 0$, $f$ is a monotone, increasing function. We set $f(0) = 0$, we have

$$\begin{aligned} f(1) &= f(0) + \frac{k}{d} \cdot f'_-\left(\frac{k}{d}\right) + \frac{1}{d} \cdot \sum_{i=0}^{d-k-1} f'_-\left(\frac{d-i}{d}\right) \\ &= 0 + \frac{k}{d} \cdot \beta \cdot \left(\frac{d-1}{d}\right)^{d-k} + \frac{1}{d} \cdot \sum_{i=0}^{d-k-1} \beta \cdot \left(\frac{d-1}{d}\right)^i \\ &= \beta \cdot \left(\frac{k}{d} \cdot \left(\frac{d-1}{d}\right)^{d-k} + 1 - \left(\frac{d-1}{d}\right)^{d-k}\right) = 1 \end{aligned}$$

Finally, we show by induction that for $i \in \{0,\dots,d-k\}$, we have $1 - f\left(\frac{d-i}{d}\right) + f'_-\left(\frac{d-i}{d}\right) = \beta$. For $i = 0$:

$$1 - f\left(\frac{d-i}{d}\right) + f'_-\left(\frac{d-i}{d}\right) = 1 - 1 + \beta \cdot \left(\frac{d-1}{d}\right)^0 = \beta$$

and for $i \in \{1,\dots,d-k\}$:

$$\begin{aligned} &1 - f\left(\frac{d-i-1}{d}\right) + f'_-\left(\frac{d-i-1}{d}\right) \\ &= 1 - f\left(\frac{d-i}{d}\right) + \frac{1}{d} \cdot f'_-\left(\frac{d-i}{d}\right) + \frac{d-1}{d} \cdot f'_-\left(\frac{d-i}{d}\right) \\ &= 1 - f\left(\frac{d-i}{d}\right) + f'_-\left(\frac{d-i}{d}\right) = \beta. \end{aligned}$$

◀

## 3.4 The algorithm's competitive ratio

We are now able to complete the proof of Theorem 2. By demonstrating that the value of the primal is at least $\tau(k,d) = \frac{1}{\beta}$ times the value of the dual solution, we conclude by the weak duality that the TSWL is $\tau(k,d)$-competitive.

We prove it by bounding the ratio of the increment of the primal and the dual objectives for every arrival of a vertex $j \in R$. We denote $\Delta P(\Delta D)$ as the primal (dual) value increment. Moreover, $\Delta D = \Delta L + \Delta R$, where $\Delta L(\Delta R)$ refers to the increment from the left side (right side). Now, we have the following key lemma:

▶ **Lemma 5.** *For every arrival of a vertex $j \in R$, we have $\Delta D \le \beta \cdot \Delta P$.*

**Proof.** Given a vertex $j \in R$, let $t = \ell(j) \cdot d$ and $t^\star = \lceil t \rceil$. We divide it into three cases:

**Case 1:** $\Delta P < 1$. Here, $t = d$ and vertex $j$'s neighbors have been fully matched, i.e., $\ell(i) = 1$ for all $i \in N(j)$. Then we would set $\Delta R = z_u = 1 - f(1) = 0$, thus we and $\Delta L \le f'_-(1) \cdot \Delta P$ since $f'$ in monotone, non-decreasing, therefore:

$$\Delta D = \Delta L + \Delta R \le f'_-(1) \cdot \Delta P + 0 = f'_-(1) \cdot \Delta P = \beta \cdot \Delta P$$

**Case 2:** $\Delta P = 1$ and $t > k$. In this case, we know that $z_j = 1 - f(t/d)$. Next, let $V_1$ be the increment volume for $j$'s neighbors between $t^\star - 1$ and $t^\star$. And Let $V_2 = 1 - V_1$, i.e., the total volume of the increment for $j$'s neighbors below $t^\star - 1$. We have

$$\begin{aligned}
\Delta L &\leq V_1 \cdot f'_-(\frac{t^\star}{d}) + V_2 \cdot f'_-(\frac{t^\star - 1}{d}) \\
&= V_1 \cdot f'_-(\frac{t^\star}{d}) + V_2 \cdot \frac{d-1}{d} \cdot f'_-(\frac{t^\star}{d}) \\
&= \frac{d-1}{d} \cdot f'_-(\frac{t^\star}{d}) + \frac{1}{d} \cdot V_1 \cdot f'_-(\frac{t^\star}{d}) \\
&\leq \frac{d-1}{d} \cdot f'_-(\frac{t^\star}{d}) + \frac{1}{d} \cdot (t - t^\star + 1) \cdot f'_-(\frac{t^\star}{d}) \\
&= f'_-(\frac{t^\star}{d}) + \frac{1}{d} \cdot (t - t^\star) \cdot f'_-(\frac{t^\star}{d})
\end{aligned}$$

where the first inequality is because $f'$ in a monotone, non-decreasing function, and the last inequality holds since we know that (note that $j \in R$, therefore $d(j) \leq d$)

$$V_1 \leq \left(\frac{t}{d} - \frac{t^\star - 1}{d}\right) \cdot d(j) \leq \left(\frac{t}{d} - \frac{t^\star - 1}{d}\right) \cdot d = t - t^\star + 1$$

And, by $f$'s definition:

$$\Delta R = 1 - f(\frac{t}{d}) = 1 - f(\frac{t^\star}{d}) - (\frac{t}{d} - \frac{t^\star}{d}) \cdot f'_-(\frac{t^\star}{d}) = 1 - f(\frac{t^\star}{d}) - \frac{1}{d} \cdot (t - t^\star) \cdot f'_-(\frac{t^\star}{d}).$$

Therefore,

$$\Delta D = \Delta R + \Delta L \leq 1 - f(\frac{t^\star}{d}) + f'_-(\frac{t^\star}{d}) = \beta.$$

**Case 3:** $\Delta P = 1$ and $t \leq k$. By the algorithm definition, $\Delta R = z_j = 1 - f(k/d)$, and $\Delta L = f'_-(k/d)$. Thus, we have

$$\Delta D = \Delta R + \Delta L = 1 - f(\frac{k}{d}) + f'_-(\frac{k}{d}) = \beta.$$ ◀

## 4 Upper Bounds for fractional matching

In this section, we provide an instance indicating that $\tau(k, d)$ is the best possible competitive ratio that can be achieved by any online algorithm.

▶ **Theorem 6.** *For fractional online matching in $(k, d)$-graphs, no online algorithm can be better than $\tau(k, d)$-competitive. This upper bound also holds for randomized algorithms.*

**Proof.** The hard instance is inspired by [3]. More concretely, we construct a parameterized family of primal linear programs based on a candidate collection of input sequences for proving the lower bound, where the objective function corresponds to optimizing the competitive ratio. Given $d, k$, our instance is a bipartite graph $G(L, R, E)$, where the number of vertices is $n = |L| = |R|$. Given a deterministic online fractional algorithm, the adversary sequence has $d$ phases in total. Wherein in the first $k$ phases, the adversary divides $L$ into $n/d$ groups, each containing $d$ vertices, and introduces a single online vertex adjacent to each group. This guarantees $d(i) \geq k$ for all $i \in L$ and that the offline can match $k$ vertices from each group. Accordingly, the adversary sets $L_{k+1} \subset L$ by dropping each group's

$k$-lowest load vertices (according to the online algorithm assignment). Next, in phase $t \in \{k+1, \ldots, d-1\} = [k+1, d-1]$, the adversary divides $L_t$ into $|L_t|/d$ groups and introduces a single online vertex adjacent to each group. The adversary defines $L_{t+1} \subset L_t$ by dropping the lowest load vertex from each group. In the last phase, for each vertex $i \in L_d$, the adversary introduces an online adjacent vertex.

Formally, the adversary defines for each phase $t \in [d]$, $L_t \subseteq L$, where all the new edges in phase $t$ would be connected only to $L_t$, and the adversarial would set $L_d \subseteq L_{d-1} \subseteq \cdots \subseteq L_1 = L$. Moreover, the adversary groups each set $L_t$ into $|L_t|/d$ disjoint groups, each of size $d$, denoted as $L_{t,s}$, for $s \in [|L_t|/d]$. In phase $t \in [d-1]$, $|L_t|/d$ online vertices arrive, and the $s^{th}$ ($s \in [|L_t|/d]$) vertex has edges to the offline vertices in the group $L_{t,s}$. For $t \in [k]$, $L_t = L$, and the adversary uses the same (arbitrary) grouping, i.e., $L_{t_1,s} = L_{t_2,s}$ for $t_1, t_2 \in [k]$, $s \in [n/d]$. According to the online algorithm assignment, the adversary sets $L_{k+1}$. Specifically, for $s \in [n/d]$, let $L^e_{k,s} \subset L_{k,s}$ the subset of the $k$-lowest load vertices in $L_{k,s}$ after phase $k$ and $L^m_{k,s} = L_{k,s} \setminus L^e_{k,s}$ (the subset of the $(d-k)$-highest load vertices in $L_{k,s}$). We define $L_{k+1} = \cup_{s=1}^{n/d} L^m_{k,s}$. note that,$|L_{k+1}| = (1 - k/d)\,|L_k|$. Similarly, the adversary sets $L_{t+1}$, for $t \in [k+1, d-1]$ as follow: For $s \in \frac{|L_t|}{d}$, let $L^e_{k,s} \subset L_{k,j}$ be the subset containing the lowest total load vertex in $L_{t,s}$ after phase $t$ and $L^m_{t,s} = L_{t,s} \setminus L^e_{t,s}$. We define $L_{t+1} = \cup_{s=1}^{|L_t|/d} L^m_{t,s}$. Note that, $|L_{t+1}| = (1 - 1/d)\,|L_t|$ for $t \in [k+1, d-1]$. Finally, in phase $d$, for each vertex $i \in L_d$, the adversary introduces an online adjacent vertex. We set $n$ as a large constant, ensuring that $|L_t|$ for $t \in [d]$ is a multiple of $d$.

Clearly, the optimal matching size is $n$ since it matches online vertices of phases $t \in [k]$ to the vertices $L^e_{k,s}$ (for $s \in [n/d]$), and for $t \in [k+1, d-1]$ match the $s$'th online vertex of phase $t$ to the vertex in $L^e_{t,s}$. The rest of the $L_d$ vertices are matched in the last phase.

## 4.1 The primal linear program

We would use $x^m_k(x^e_k)$ as the average load assigned in phases $[k]$ on vertices in $L^m_{k,s}(L^e_{k,s})$ for $s \in [n/d]$. Similarly, for $t \in [k+1, d-1]$ we use $x^m_t(x^k_t)$ as the average load assigned in phase $t$ to vertices in $L^m_{t,s}(L^e_{t,s})$. For $t \in [k+1, d-1]$, we use $u^m_t(u^e_k)$ as the average total load assigned before phase $t$ to vertices in $L^m_{t,s}(L^e_{t,s})$, and $u^m_d$ as the average total load assigned before phase $d$ to $L_d$.

$$\textbf{max } \tfrac{|L_k|}{d}((d-k)x^m_k + k \cdot x^e_k) + \sum_{t=k+1}^{d-1} \frac{|L_t|}{d}(x^e_t + (d-1)x^m_t) + |L_d|(1 - u^m_d)$$

$$(d-k) \cdot x^m_k + k \cdot x^e_k \le k \qquad (v_k)$$

$$(d-1) \cdot x^m_t + x^e_t \le 1 \qquad \forall t \in [k+1, d-1], \quad (v_t)$$

$$x^e_k \le x^m_k \qquad (m_k)$$

$$x^e_t + u^e_t \le x^m_t + u^m_t \qquad \forall t \in [k+1, d-1], \quad (m_t)$$

$$d \cdot x^m_k \le (d-1) \cdot u^m_{k+1} + u^e_{k+1} \qquad (c_k)$$

$$d \cdot (x^m_t + u^m_t) \le (d-1) \cdot u^m_{t+1} + u^e_{t+1} \qquad \forall t \in [k+1, d-2], \quad (c_t)$$

$$d \cdot (x^m_{d-1} + u^m_{d-1}) \le d \cdot u^m_d \qquad (c_{d-1})$$

$$x^m_t, u^m_t, x^e_t, u^e_t \ge 0 \qquad \forall t \in [k, d]$$

Constraint $v_k$ implies that the total volume in vertices in $L^m_k$ (note, $\frac{|L^m_k|}{|L_k|} = \frac{d-k}{d}$) plus the total volume of vertices in $L^e_k$ (note, $\frac{|L^e_k|}{|L_k|} = \frac{k}{d}$) does not exceed the total volume in the first $k$ phases. Constraint $v_t$ for $t \in [k+1, d-1]$ implies that the total volume of vertices

in $L_t^m$ (note, $\frac{|L_t^m|}{|L_t|} = \frac{d-1}{d}$) plus the total volume of vertices in $L_t^e$ (note, $\frac{|L_t^e|}{|L_t|} = \frac{1}{d}$) does not exceed the total volume assigned in those phases. Constraints $m_t$ for $t \in [k, d-1]$ preserve the monotonicity property, i.e., the average total load of vertices in $L_{t,s}^m$ is higher than the average total load of vertices in $L_{e,s}^m$, by the groups' definition. Constraints $c_t$ for $t \in [k, d-1]$ preserve the total volume of $L_t^m$ after phase $t$ as the total volume of $L_{t+1}$ prior to phase $t+1$. Finally, the objective function captures the fractional volume assigned on $L$.

## 4.2 The dual linear program

The dual of the linear program is:

$$\begin{aligned}
&\textbf{min}\; k \cdot v_k + \sum_{t=k}^{d-1} v_t + |L_d| \\
&(d-k) \cdot v_k - m_k + d \cdot c_k \geq n \cdot (1 - k/d) && (x_k^m)\\
&k \cdot v_k + m_k \geq \frac{n \cdot k}{d} && (x_k^e)\\
&(d-1) \cdot v_t - m_t + d \cdot c_t \geq n \cdot (1 - k/d) \cdot (1 - 1/d)^{t-k} \qquad \forall t \in [k+1, d-1], && (x_t^m)\\
&v_t + m_t \geq \frac{n}{d} \cdot (1 - k/d) \cdot (1 - 1/d)^{t-k-1} \qquad \forall t \in [k+1, d-1], && (x_t^m)\\
&-m_t - (d-1) \cdot c_{t-1} + d \cdot c_t \geq 0 \qquad \forall t \in [k+1, d-1], && (u_t^m)\\
&m_t - c_{t-1} \geq 0 \qquad \forall t \in [k+1, d-1], && (u_t^e)\\
&-d \cdot c_{d-1} \geq -n \cdot (1 - k/d) \cdot (1 - 1/d)^{d-k-1} && (u_d^m)
\end{aligned}$$

## 4.3 Constructing the dual solution

By assuming the constraints are tight, we have by constraint $(u_d^m)$: $c_{d-1} = \frac{n}{d} \cdot (1 - 1/d)^{d-k-1}$, by summing constraints $(u_t^m)$ and $(u_t^e)$ for all $t \in [k+1, d-1]$: $c_t = c_{t-1}$, and, therefore,

$$c_t = \frac{n}{d} \cdot (1 - 1/d)^{d-k-1}, \text{ for all } t \in [k, d-1].$$

By summing constraints $(x_k^m)$ and $(x_k^e)$ for all $t \in [k+1, d-1]$: $d \cdot v_k + d \cdot c_k = n$, therefore:

$$v_k = n \cdot \frac{1 - d \cdot c_k}{d} = \frac{n}{d} \cdot \left(1 - (1 - k/d) \cdot (1 - 1/d)^{d-k-1}\right).$$

By summing constraints $(x_t^m)$ and $(x_k^e)$ for all $t \in [k+1, d-1]$, we have: $d \cdot v_t + d \cdot c_t = n \cdot (1 - k/d) \cdot (1 - 1/d)^{t-k-1}$, therefore:

$$v_t = \frac{n}{d} \cdot (1 - k/d) \cdot (1 - 1/d)^{d-k-1} \left((1 - 1/d)^{t-d} - 1\right), \text{ for all } t \in [k+1, d-1].$$

Finally, by constraint $x_k^e$, $m_k = \frac{n \cdot k}{d} - k \cdot v_k$, and by constraint $x_t^m$, $m_t = \frac{n}{d} \cdot (1 - k/d) \cdot (1 - 1/d)^{t-k-1} - v_t$. It is easy to verify that the dual is feasible. The value of the dual objective function:

$$\begin{aligned}
k \cdot v_k + \sum_{t=k+1}^{d-1} v_t + |L_d| &= \frac{n \cdot k}{d} \cdot \left(1 - (1 - k/d) \cdot (1 - 1/d)^{d-k-1}\right) \\
&\quad + \sum_{t=k+1}^{d-1} \frac{n}{d} \cdot (1 - k/d) \cdot (1 - 1/d)^{d-k-1} \left((1 - 1/d)^{t-d} - 1\right) \\
&\quad + n \cdot (1 - k/d) \cdot (1 - 1/d)^{d-k-1} \\
&= n \cdot \tau(k, d)
\end{aligned}$$

Therefore, there is a feasible solution for the dual, with the value $n \cdot \tau(k,d)$. By weak duality, and since the primal captures the objective of the online algorithm, any online algorithm will fractionally match at most $n \cdot \tau(k,d)$ vertices. Finally, since by our construction, the optimal value is $n$, any online is at most $\tau(k,d)$ competitive as required. See discussion in [3] for applying the lower bound also on randomized algorithms. ◀

# 5 Allocation problem

We improve the competitive ratio for the *allocation problem* on $(k,d)$-graphs for $k \leq d$.

## The allocation problem

In the allocation problem, a seller is interested in selling products to a group of buyers $L = [n]$, where buyer $i \in L$ has a budget $B(i)$, and $R$ is a set of products the seller introduces one by one. Each product $j \in R$ has a fixed price $b(j)$. Upon the arrival of a product, the buyers announce to the seller whether they are interested in buying the current product for the set price. The seller then decides (instantly) which of the interested buyers to sell the product to. Using the AdWords formulation, we have $b(i,j) \in \{0, b(j)\}$ for all $i$.

For $k \geq d$, there exists a trivial online fractional solution. Assume $k \leq d$ and let $\alpha = k/d$ and $\tau(\alpha) = 1 - (1-\alpha) \cdot e^{\alpha - 1}$, we will show that there exists a $\tau(\alpha)$-competitive fractional online algorithm.

▶ **Theorem 7.** *For the fractional online allocation problem in $(k,d)$-graphs, where $\alpha = k/d$, there exists a $\tau(\alpha)$-competitive algorithm.*

Note that the upper bound of Section 4 also holds for the allocation problem; hence, the result is tight. The algorithm and analysis are similar to the upper bound in Section 4, except we use a smooth potential function $f^{(\alpha)}$.

Let $\alpha = k/d$, $\tau(\alpha) = 1 - (1-\alpha) \cdot e^{\alpha-1}$, and for a fixed alpha, set $\beta = 1/\tau(\alpha)$, and define $g(x) = 1 + \beta \cdot (\exp(x-1) - 1)$, and $f^{(\alpha)}$,

$$f^{(\alpha)}(x) = \begin{cases} g(x) & \text{for } x \in [\alpha, 1] \\ \frac{x \cdot g(\alpha)}{\alpha} & \text{for } x \in [0, \alpha), \end{cases}$$

▶ **Lemma 8.** *For $f = f^{(\alpha)}$:*

- $f(0) = 0$, $f(1) = 1$.
- *$f$ and $f'$ are continuous functions in the segment $[0,1]$, and $f, f'$ are monotone non-decreasing functions.*
- *For $x \in [\alpha, 1]$ we have $1 - f(x) + f'(x) = \beta$*

**Proof.** Clearly, $f(x), f'(x)$ are continuous functions in the segments $[0,\alpha), (\alpha, 1]$. In addition, $f_-(\alpha) = g(\alpha) = f_+(\alpha)$, so $f$ is a continuous function and

$$\begin{aligned} f'_+(\alpha) &= g'(\alpha) = \beta \cdot \exp(\alpha - 1) \\ &= \frac{\beta \cdot \alpha \exp(\alpha-1)}{\alpha} = \frac{\beta \cdot (1/\beta - 1 + \exp(\alpha-1))}{\alpha} = \frac{1 + \beta \cdot (\exp(\alpha-1) - 1)}{\alpha} \\ &= \frac{g(\alpha)}{\alpha} = f'_-(\alpha), \end{aligned}$$

therefore, $f'$ is continuous as well. And $f'(x) > 0$ for $x \in [0,1]$. Moreover, $f''(x) = 0$ for $x \in [0,\alpha]$ and $f''(x) = \exp(x-1)/\beta \geq 0$; hence, $f'$ is monotone non-decreasing. $f(0) = 0 \cdot g(\alpha)/\alpha = 0$, and $f(1) = g(1) = 1 + \beta \cdot (\exp(1-1) - 1) = 1$. Finally, for $x \in [\alpha, 1]$

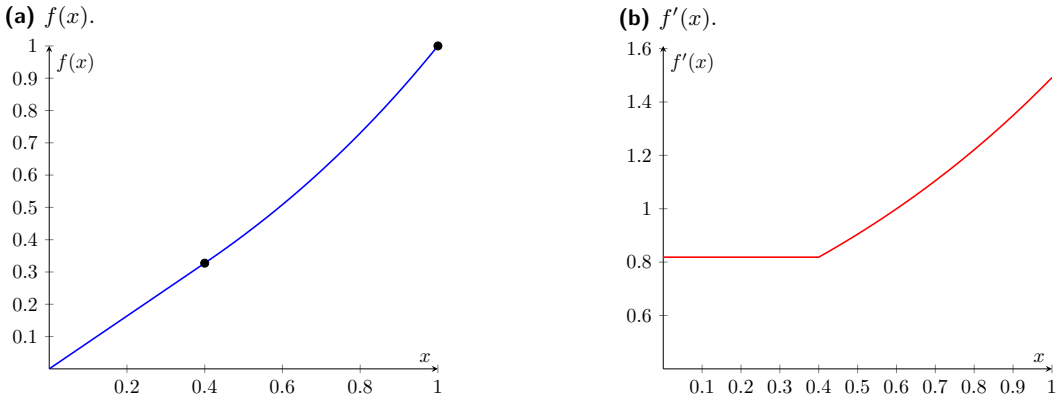



**Figure 4** The functions $f^{(\alpha)}$ and $f'^{(\alpha)}$, for $\alpha = 0.4$.

$$1 - f(x) + f'(x) = 1 - g(x) - g'(x)$$
$$= 1 - (1 + \beta \cdot (\exp(x-1) - 1)) + \beta \cdot \exp(x-1) = \beta. \quad \blacktriangleleft$$

## 5.1 The Two-Step-Water-Level Allocation algorithm

For $i \in L$, let $N(i)$ be the set of products that buyer $i$ is interested in. Let $T(i) = \frac{\sum_{j \in N(i)} b(j)}{B(i)}$ be the ratio between the sum of prices in which buyer $i$ is interested, to its budget, and let $G(i) = \sum_{j \in N(i)} b(j) \cdot x_{i,j}$ be the total gain of bidder $i$, the level of buyer $i \in L$, defined as $\ell(i) = \frac{G(i)}{B(i)}$. Accordingly, we define the degree and level after the arrival of product $j$, as $T^j(i) = \sum_{t \in N(i), t \leq j} b(t)$, and $\ell^j(i) = \frac{G^j(i)}{B(i)}$, where $G^j(i) = \sum_{t \in N(i), t \leq j} b(t) \cdot x_{i,t}$.

**Algorithm 2** The **Two-Step-Water-Level Allocation** algorithm.

Upon the arrival of a new product $j \in R$:

1. For each buyer $i \in N(j)$, such that $T^j(i) \leq k$
   Increase (if necessary) $G^j(i)$, the gain of buyer $i$, to $T^j(i)/d$
2. Using the leftover volume of product $j$:
   Increase the gain of bidders in $N(j)$ using the water-level algorithm (according to the buyer's level)
3. Update the dual variables, let $\ell(j) = \min_{i \in N(j)} \ell^j(i)$ and $\hat{\ell}(j) = \max(k/d, \ell(j))$.
   Set $y_i = f^{(\alpha)}(\ell(i))$, for $i \in N(j)$
   Set $z_j = 1 - f^{(\alpha)}(\hat{\ell}(j))$

## 5.2 The algorithm's feasibility

First, we will prove that for a proper $f^{(\alpha)}$ (for brevity's sake, we use $f$ for $f^{(\alpha)}$ for the rest of the section), the primal and the dual solutions of **Two-Step-Water-Level Allocation algorithm** (TSWLA) are feasible.

▶ **Lemma 9.** *Given a function $f : [0,1] \to [0,1]$, where $f$ is monotone, non-decreasing the primal and dual solutions of the TSWLA algorithm are feasible at the end of the algorithm.*

**Proof.** First, we will show that the algorithm does not over-allocate product $j$ in step 1. We assume that the algorithm is feasible until the arrival of product $j$, and we bound the value of $x_{i,j}$ after step 1, and we will show that $x_{i,j} \leq 1/d$, for all $i \in N(j)$. For $i \in N(j)$, by our assumption that the algorithm is feasible until the arrival of product $j$, its level before the arrival of $j$ is at least $\frac{T^j(i)-b(j)/B(i)}{d}$, by rearranging the terms $x_{i,j} \leq \frac{1}{d}$. Second, $\sum_{i \in N(j)} x_{i,j} \leq \frac{|N(j)|}{d} \leq 1$, by $d$'s definition. In the second step, the water level algorithm never increases the level above one, and, therefore, $G(i) \leq B(i)$ for all $i \in L$. We will show that for any monotone, non-decreasing function $f : [0,1] \to [0,1]$, the dual is feasible at the end of the algorithm's run. We have for all $i \in L$, $\ell(i) \geq \frac{\min(T(i),k)}{d} \geq k/d$, where the last inequality is by $k$'s definition. The dual constraints: If $\ell(j) \leq k/d$, for $i \in N(j)$, we have

$$b(j) \cdot y_i + z_j = b(j) \cdot f(\ell(i)) + b(j) \cdot (1 - f(k/d)) \geq b(j),$$

since $\ell(i) \geq k/d$ for all $i \in L$ at the end of the algorithm and $f$ is monotone, non-decreasing. If $\ell(j) > k/d$, we have,

$$b(j) \cdot y_i + z_j \geq b(j) \cdot f(\ell^j(i)) + b(j) \cdot (1 - f(\hat{\ell}(j))) \geq b(j),$$

where the first inequality is because $\ell(i)$ only increases, and $f$ is monotone, non-decreasing. ◀

## 5.3 Algorithm's competitive ratio

We are now able to complete the proof of Theorem 7. By proving that the value of the Primal is at least the $\tau(\alpha) = \frac{1}{\beta}$ times the value of the dual solution, then, by weak duality, we will conclude the TSWLA is $\tau(\alpha)$-competitive as required.

We prove it by bounding the ratio of the increment of the primal and the dual for every arrival. We denote $\Delta P(\Delta D)$ to denote the increment for the primal (dual) variable. Moreover, $\Delta D = \Delta L + \Delta R$, where $\Delta L(\Delta R)$ refers to the increment from the left side (right side). Now, we have the following key lemma:

▶ **Lemma 10.** *For every arrival of a product $j \in R$, we have $\Delta D \leq \Delta P \cdot \beta$.*

**Proof.** At every step $t$, we have $\Delta R = z_t$, $\Delta L = \sum_{i \in N(j)} \Delta L_i$, where $\Delta L_i = B(i) \cdot (y_i^j - y_i^{j-1})$

We note that, $\frac{d\Delta P_i}{dx_{i,j}} = b(j)$. Therefore, $\Delta P = b(j) \cdot \sum_{i \in N(j)} x_{i,j}$.

$$\frac{d\Delta L_i}{dx_{i,j}} = B(i) \cdot \frac{dy_i}{dx_{i,j}} = B(i) \cdot \frac{b(j)}{B(i)} \cdot f'(\ell(i)) \leq b(j) \cdot f'(\ell^j(i)) = b(j) \cdot f'(\hat{\ell}(j))$$

where the inequality is because $f'$ is a monotone non-decreasing function, and the last equality is since $f'(x) = f'(\alpha)$ for $x \in [0, \alpha]$.

Therefore, we have $\Delta L \leq b(j) \cdot f'(\hat{\ell}(j)) \sum_{i \in N(u)} x_{i,j}$

**Case 1.** $\sum_i x_{i,j} < 1$. In this case, $\ell(j) = 1$, $\Delta R = z_j = 1 - f(1) = 0$, and we have

$$\Delta D = \Delta L + \Delta R \leq b(j) \cdot f'(1) \cdot \sum_i x_{i,j} = (1 - f(1) + f'(1)) \cdot \Delta P = \beta \cdot \Delta P$$

**Case 2.** $\sum_i x_{i,j} = 1$, $\ell(j) > k/d$. We have $\Delta R = z_j = b(j) \cdot (1 - f(\ell(j))$ and $\Delta P = b(j)$.

$$\Delta D = \Delta L + \Delta R \leq b(j) \cdot f'(\ell(j)) + b(j) \cdot (1 - f(\ell(j)) = b(j) \cdot (1 - f(\ell(j)) + f'(\ell(j))) = \beta \cdot \Delta P$$

**Case 3.** $\sum_i x_{i,j} = 1$, $\ell(j) \leq k/d$. Again, we have $\Delta P = b(j)$. We know that $\Delta R = z_j = b(u) \cdot (1 - f(k/d))$. Note that $f'(x) = f'(k/d)$ for $x \in [0, k/d]$.

$$\Delta D = \Delta L + \Delta R \leq b(j) \cdot f'(\ell(j)) + b(j) \cdot (1 - f(k/d)) = b(j) \cdot (1 - f(k/d) + f'(k/d)) = \beta \cdot \Delta P.$$ ◀

# 6 Upper Bounds for the Adwords problem

In this section, we prove that $\tau(k,d)$ competitiveness is impossible for the Adwords problem.

▶ **Theorem 11.** *For fractional AdWords in $(k,d)$-graphs, there exists $k,d$ and fixed $\epsilon > 0$, such that no online algorithm can be better than $(\tau(k,d)-\epsilon)$-competitive.*

The hard instance is composed of several scenarios. The intuition behind the scenario is that if for a certain $j$, $b(i_1,j) > b(i_2,j)$, then, on the one hand, the online algorithm should allocate the item to $i_1$ in order not to lose its higher price, and, on the other hand, by doing so the optimal allocation can use item $j$ to $i_2$. We amplify this example and prove an improved upper bound for $\alpha = 1/2$ ($k = d/2$) and $B(i) = 1$ for all $i \in L$. First, we bound the gain of a subset of the bidders after several sequence steps.

## 6.1 Bounding the gain of a subset of the bidders

Let $Q \subseteq L$ be a subset of vertices. After several sequence steps, we bound the total online gain that can be extracted from $Q$, where the degree of $i \in Q$ is already $k$, and the adversary determined how many bidders from $Q$ have already exhausted their budget in previous steps. We compute an upper bound on the total gain of an online algorithm on $Q$ as a function of the average online gain on $Q$ in previous steps by defining the rest of the sequence for this subset $Q$ and bounding the total gain using a linear program. The rest of the sequence for this $Q$ is as the second part of Section 4. Specifically, it defines $Q_{k+1}$ as a subset of $Q$ in which the current gain in OPT is 0. In phase $t \in [k+1, d-1]$, the adversary divides $Q_t$ into $|Q_t|/d$ groups and introduces for each group's bidders a product with an equal bid value of 1. The adversary defines $Q_{t+1}$ by dropping the lowest gain bidder from each group. In the last phase, for each bidder $i \in Q_d$, the adversary introduces a product with a bid of 1.

Formally, given such $Q$ and the decomposition of $Q$ into $Q^e_k, Q^m_k$ where the average gain in $Q^e_k$ is, at most, the average gain of $Q^m_k$ For $t \in [k+1,d]$, the adversarial would define at each phase $t \in [k+1,d]$, $Q_t \subseteq Q$, where $Q_{k+1} = Q^m_k$. The adversary groups each $Q_t$ into $|Q_t|/d$ disjoint groups, each of size $d$, denoted as $Q_{t,s}$, for $s \in [|Q_t|/d]$. In phase $t \in [k+1,,d-1]$, $|Q_t|/d$ products arrive and the $s$'th ($s \in [|Q_t|/d]$) bid $b(i,s) = 1$ to $i \in Q_{t,s}$ (and 0 otherwise). In phase $t \in [k+1,,d-1]$, $|Q_t|/d$ products arrive and the $s$'th ($s \in [|Q_t|/d]$) product bids are $b(i,s) = 1$ to all the bidders in group $i \in Q_{t,s}$. $Q_{t+1}$ for $t \in [k+1,d-1]$ is set as follows: For $s \in \frac{|Q_t|}{d}$, let $Q^e_{k,s} \subset Q_{k,s}$ be the subset containing the lowest gain bidder in $Q_{t,s}$ after phase $t$ and $Q^m_{t,s} = Q_{t,j} \setminus Q^e_{t,j}$. We define $Q_{t+1} = \cup_{s=1}^{|Q_t|/d} Q^m_{t,s}$. note that, $|Q_{t+1}| = (1-1/d)\,|Q_t|$ for $t \in [k+1,d-1]$. Let $G_Q$ be the total gain of $Q$ vertices.

**Bounding $G_Q$ as a function of previous phases.** Next, we bound the total gain of $G_Q$, given that after phase $k$, a decomposition of $Q$ into $Q^e_k, Q^m_k$ exists (i.e., $Q = Q^e_k \cup Q^m_k$) such that $x^m_k$ ($x^e_k$) is the average load on $Q^m_k$ ($Q^e_k$) and $x^e_k \le x^m_k$ (constraint $\tilde{m}_k$). Assuming that the total gain of $Q$ until (not including) step $k+1$ is $G^k_Q$, we have $\frac{|Q^m_k|}{|Q|} \cdot x^m_k + \frac{|Q^e_k|}{|Q|} \cdot x^e_k \le \frac{G^k_Q}{|Q|}$, constraint $\tilde{v}_k$). We will define $Q_{k+1} = Q^m_k$, and accordingly, we have $d \cdot x^m_k \le (d-1) \cdot u^m_{k+1} + u^e_{k+1}$ (constraint $\tilde{n}_k$). Denote $V^Q = \frac{G^Q_k}{|Q|}$, and $r^Q = \frac{|Q^m_k|}{|Q|}$ the following Linear program, will bound $G(V^Q, r^Q)$ the total gain that can be extracted from $Q$.

$G(V^Q, r^Q) = \mathbf{max}\, |Q| \cdot V^Q + \sum_{t=k+1}^{d-1} \frac{|Q_t|}{d} \cdot (x_t^e + (d-1)x_t^m) + |Q_d| \cdot (1 - u_d^m)$,
such that:

$$
\begin{aligned}
& r^Q \cdot x_k^m + (1 - r^Q) \cdot x_k^e \leq V^Q && && (\tilde{v}_k)\\
& (d-1) \cdot x_t^m + x_t^e \leq 1 && \forall t \in [k+1, d-1], && (v_t)\\
& x_k^e \leq x_k^m && && (\tilde{m}_k)\\
& x_t^e + u_t^e \leq x_t^m + u_t^m && \forall t \in [k+1, d-1], && (m_t)\\
& d \cdot x_k^m \leq (d-1) \cdot u_{k+1}^m + u_{k+1}^e && && (\tilde{n}_k)\\
& d \cdot (x_t^m + u_t^m) \leq (d-1) \cdot u_{t+1}^m + u_{t+1}^e && \forall t \in [k+1, d-2], && (c_t)\\
& d \cdot (x_{d-1}^m + u_{d-1}^m) \leq d \cdot u_d^m && && (c_{d-1})\\
& x_t^m, u_t^m, x_t^e, u_t^e \geq 0 && && \forall t \in [k, d]
\end{aligned}
$$

## 6.2 The Scenarios

Let $L = U \cup D$, such that $|U| = |D| = n$, We divide $U(D)$ into $n/k$ disjoint groups $U_{k,s}(D_{k,s})$ such that $|U_{k,s}| = |D_{k,s}| = k$, for $s \in [n/k]$.

**Phase 0.** For each $s \in [n/k]$, introduce a product $j$, such that $b(i,j) = 0.26 \cdot d$ for $i \in U_{k,s}$ and $b(i,j) = 0.24 \cdot d$ for $j \in D_{k,s}$. Any online algorithm must determine $\gamma = \sum_{i \in U_{k,s}, j} x_{i,j}/n$, the average portion of the items of phase 0 assigned to $U$. Next, the adversary introduces products of phase $k$ (phases $[k-1]$ are empty).

**First Scenario.** For each $s \in [n/(2 \cdot k)]$, the adversary introduces two products $j_1, j_2$, such that $b(i, j_1) = 0.24 \cdot d$ for $i \in U_{k,2 \cdot s-1} \cup U_{k,2 \cdot s}$ and $b(i, j_2) = 0.26 \cdot d$ for $i \in D_{k,2 \cdot s-1} \cup D_{k,2 \cdot s}$.

**Second Scenario.** For each $s \in [n/k]$, the adversary introduces a product $j$, such that $b(i,j) = 0.24 \cdot d$ for $i \in U_{k,s}$ and $b(i,j) = 0.26 \cdot d$ for $j \in D_{k,s}$. Any online algorithm must determine $\delta = \sum_{i \in U_{k,s}, j} x_{i,j}/n$, the average portion of the items of phase $k$ of the second scenario assigned to $U$. Note that, in both scenarios, the current degree of each vertex $i \in L$ is $k$. Using the values $\gamma, \delta$, we bound the average gain per bidder of $U(D)$ up to phase $k$ for scenario $o$, denoted as $V^{U(o)}(V^{D(o)})$. For the first scenario:

$$
\begin{aligned}
V^{U(1)} =& \frac{1}{n} \cdot \left( \frac{n}{k} \cdot \gamma \cdot 0.26 \cdot d + \frac{n}{d} \cdot 0.24 \cdot d \right) = 0.52 \cdot \gamma + 0.24\\
V^{D(1)} =& \frac{1}{n} \cdot \left( \frac{n}{k} \cdot (1-\gamma) \cdot d \cdot 0.24 \cdot d + \frac{n}{d} \cdot 0.26 \cdot d \right) = 0.74 - 0.48 \cdot \gamma
\end{aligned}
$$

For the second scenario:

$$
\begin{aligned}
V^{U(2)} =& \frac{1}{n} \cdot \left( \frac{n}{k} \cdot \gamma \cdot 0.26 \cdot d + \frac{n}{k} \cdot \delta \cdot 0.24 \cdot d \right) = 0.52 \cdot \gamma + 0.48 \cdot \delta\\
V^{D(2)} =& \frac{1}{n} \cdot \left( \frac{n}{k} \cdot (1-\gamma) \cdot d) \cdot 0.24 \cdot d + \frac{n}{k} \cdot (1-\delta) \cdot 0.26 \cdot d \right) = 1 - 0.48 \cdot \gamma - 0.52 \cdot \delta
\end{aligned}
$$

Similarly, we denote $\gamma^{\mathrm{OPT}}, \delta^{\mathrm{OPT}}$ as the corresponding values for the optimal allocation. After setting those values (for a certain case), the gain of OPT on this subset would be determined. Then, the adversary can omit bidders in the corresponding subset, i.e., determining $r^Q$ for $Q \in U, D$.

**Continuing the first scenario.** In the first scenario, the adversary sets $\gamma^{\text{OPT}} = 0$ (i.e., uses the products of phase 0 to increase the gain of vertices only in $D$). Taking into account also the products of phase $k$ of the first scenario, it omits the $r^U = 0.24$ portion from $U$ and the $r^D = 0.74$ portion from $D$, and continues the sequence for $U, D$ separately using Subsection 6.1 construction.

Formally, for $s \in [n/k]$, let $U^e_{k,j} \subset U_{k,s}$ be the subset of the $(r^U \cdot k)$-lowest gain vertices in $U_{k,s}$ after phase $k$ and $U^m_{k,s} = U_{k,s} \setminus U^e_{k,s}$ (the subset of the $((1-r^U) \cdot k)$-highest load vertices in $U_{k,j}$), and define $U_{k+1} = \cup_s U^m_{k,s}$. Accordingly, for $s \in [n/k]$, let $D^e_{k,s} \subset D_{k,s}$ be the subset of the $(r^D \cdot k)$-lowest gain vertices in $U_{k,s}$ after phase $k$ and $U^m_{k,j} = U_{k,j} \setminus Q^e_{k,j}$ (the subset of the $((1-r^D) \cdot k)$-highest load vertices in $Q_{k,j}$). We continue the scenario separately for $U$ and $D$ using Subsection 6.1 construction. We bound the total gain of this scenario using the LP $G(V^Q, r^Q)$. Specifically, the total gain is at most (as a function of $\gamma$):

$$G(V^U, r^U) + G(V^D, r^D) = G(0.52 \cdot \gamma + 0.24, 0.24) + G(0.74 - 0.48 \cdot \gamma, 0.74).$$

**Continuing the second scenario.** In the second scenario, the adversary uses $\gamma^{\text{OPT}} = 1$, $\delta^{\text{OPT}} = 0$ (i.e., uses the products of phase 0 to increase the gain of vertices only in $U$ and the products of the second scenario of phase $k$ to increase the gain of vertices only in $D$). In this case, it omits the $r^U = 0.52$ portion from $U$ and the $r^D = 0.52$ portion from $D$. Similarly to the previous case, we have:

$$G(V^U, r^U) + G(V^D, r^D) = G(0.52 \cdot \gamma + 0.48 \cdot \delta, 0.52) + G(1 - 0.48 \cdot \gamma - 0.52 \cdot \delta, 0.52).$$

By our construction definition, in all scenarios, the optimal value is $2 \cdot n$. Therefore, in order to bound $c$, we have the following $LP$

$$\begin{aligned} &\textbf{max}\, c \\ \text{such that: } &G(0.52 \cdot \gamma + 0.24, 0.24) + G(0.74 - 0.48 \cdot \gamma, 0.74) \geq 2 \cdot n \cdot c \\ &G(0.52 \cdot \gamma + 0.48 \cdot \delta, 0.52) + G(1 - 0.48 \cdot \gamma - 0.52 \cdot \delta, 0.52) \geq 2 \cdot n \cdot c \end{aligned}$$

By solving it numerically, we have for $d = 100$, $k = 50$, we have:

$c \leq 0.69485 < \tau(0.5) \approx 0.6967$.

## References


1. Susanne Albers and Sebastian Schubert. Online ad allocation in bounded-degree graphs. In *Web and Internet Economics: 18th International Conference, WINE 2022, Troy, NY, USA, December 12–15, 2022, Proceedings*, pages 60–77. Springer, 2022.
2. Susanne Albers and Sebastian Schubert. Tight bounds for online matching in bounded-degree graphs with vertex capacities. In *ESA*, volume 244 of *LIPIcs*, pages 4:1–4:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
3. Yossi Azar, Ilan Reuven Cohen, and Alan Roytman. Online lower bounds via duality. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1038–1050. SIAM, 2017.
4. Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *Algorithms–ESA 2010: 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I 18*, pages 170–181. Springer, 2010.
5. Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms–ESA 2007: 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007. Proceedings 15*, pages 253–264. Springer, 2007.

6 Niv Buchbinder, Joseph Seffi Naor, et al. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

7 Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and Shan Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 117–126. IEEE, 2009.

8 Bala Kalyanasundaram and Kirk R Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1-2):319–325, 2000.

9 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 587–596, 2011.

10 Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.

11 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

12 Vahideh H Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. *Mathematics of Operations Research*, 37(4):559–573, 2012.

13 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22–es, 2007.

14 Joseph Naor and David Wajc. Near-optimum online ad allocation for targeted advertising. *ACM Transactions on Economics and Computation (TEAC)*, 6(3-4):1–20, 2018.

# Robust and Space-Efficient Dual Adversary Quantum Query Algorithms

**Michael Czekanski**
Department of Statistics and Data Science, Cornell University, Ithaca, NY, USA

**Shelby Kimmel**
Department of Computer Science, Middlebury College, VT, USA

**R. Teal Witter**
Department of Computer Science and Engineering, New York University, Brooklyn, NY, USA

## Abstract

The general adversary dual is a powerful tool in quantum computing because it gives a query-optimal bounded-error quantum algorithm for deciding any Boolean function. Unfortunately, the algorithm uses linear qubits in the worst case, and only works if the constraints of the general adversary dual are exactly satisfied. The challenge of improving the algorithm is that it is brittle to arbitrarily small errors since it relies on a reflection over a span of vectors. We overcome this challenge and build a robust dual adversary algorithm that can handle approximately satisfied constraints. As one application of our robust algorithm, we prove that for any Boolean function with polynomially many 1-valued inputs (or in fact a slightly weaker condition) there is a query-optimal algorithm that uses logarithmic qubits. As another application, we prove that numerically derived, approximate solutions to the general adversary dual give a bounded-error quantum algorithm under certain conditions. Further, we show that these conditions empirically hold with reasonable iterations for Boolean functions with small domains. We also develop several tools that may be of independent interest, including a robust approximate spectral gap lemma, a method to compress a general adversary dual solution using the Johnson-Lindenstrauss lemma, and open-source code to find solutions to the general adversary dual.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Theory of computation → Quantum query complexity

**Keywords and phrases** Quantum Computing, Robust Quantum Algorithms, Johnson-Lindenstrauss Lemma, Span Programs, Query Complexity, Space Complexity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.36

**Related Version** *Full Version*: `https://arxiv.org/abs/2306.15040` [17]

**Supplementary Material**
*Software (Source Code)*: `https://github.com/rtealwitter/QuantumQueryOptimizer`
archived at `swh:1:dir:60941e1654f097fdcaf2c413c8f7955482f84eb1`

**Funding** *Shelby Kimmel*: sponsored by the U.S. Army Research Office and this work was accomplished under Grant Number W911NF-20-1-0327. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
*R. Teal Witter*: supported by NSF No. 1922658 and 1916505.

**Acknowledgements** We thank Stacey Jeffery for elucidating discussions.




## 1 Introduction

The query model of computation has proven a powerful model in which to prove quantum-classical separations [26, 19, 15] and to understand the limits of quantum algorithms [2, 6, 28]. The power and usefulness of this computational model in the quantum setting in part derives from the fact that bounded-error quantum query complexity for Boolean function evaluation is tightly and beautifully described by a semidefinite program (SDP) – the general adversary bound [46, 44]. The dual of this semidefinite program has played an important role in the development and understanding of quantum algorithms. In particular, the dual has been used to show the optimality of span program algorithms, which are a critical element for several algorithm design paradigms [45, 8, 30, 32] and are useful in a wide range of applications [11, 20, 12, 9, 7]. While we have methods to create a bounded-error quantum algorithm for function evaluation based on a set of vectors that exactly satisfy the constraints of the general adversary dual [44], it is natural to ask how robust this algorithm is to transformations and perturbations. For example, can we take a vector set that satisfies the dual adversary constraints and modify it to obtain an algorithm with better space complexity? Or can we create an algorithm from a vector set that satisfies relaxed dual adversary constraints? In this paper, we find criteria under which these modified or approximately satisfying vector sets yield viable algorithms, and we consider two problems where such robustness is useful.

First, we study when we can reduce the space complexity of dual adversary-derived algorithms. Almost all Boolean functions on $n$ bits have unitary space complexity $\Omega(n)$ [31], but we hope to discover conditions under which less space might be required. We do this by determining when we can compress the dimension of a set of vectors that exactly satisfies the dual adversary constraints. While the dual adversary is most commonly studied in the context of query complexity, is it also closely related to unitary space complexity [31], and the dimension of the satisfying vectors to the dual adversary problem determines the space used by the algorithm. As building large quantum computers will likely continue to be a technical challenge in the near to medium term [29], finding ways to minimize the space used by quantum computers is important.

We use two approaches to compress the dimension of a satisfying vector set, and hence reduce the space used by the resulting algorithm. First, we consider using a unitary transformation to rotate the vectors to a smaller space. Next, we analyze applying the Johnson-Lindenstrauss (JL) lemma, which is a powerful tool for compressing the dimension of a vector set while approximately preserving the structure of the vectors. With our analysis, we find the simpler, unitary transformation always results in a better compression than the JL approach. But, with either compression method, we can show for any function with polynomially many 1-valued inputs, or polynomially many 0-valued inputs, there is a query-optimal algorithm that uses logarithmic space. While it is not surprising that there is an algorithm that uses logarithmic space for such problems (one could iterate through possible 1/0-valued inputs and run Grover's search to test each one), it is not obvious that there is a *query-optimal* algorithm that uses logarithmic space.

The second problem we consider is how to create an algorithm using the output of a numerical SDP solver applied to the general adversary dual. One can plug the general adversary dual into a classical SDP solver and find a set of vectors that is close to a query-optimal, exactly satisfying vector set. Due to finite precision, we expect that the numerical solution will almost never be *exactly* satisfying, but we would like to know if we can still produce a query-optimal quantum algorithm. With our tools for creating robust dual adversary algorithms, we bound the error that can be tolerated, and we describe how

to take a vector set that does not quite satisfy the dual adversary constraints, but is within the tolerable error, and use it to create a bounded-error algorithm. In the worst case, our analysis requires an error that scales inversely with the number of 1-valued or 0-valued inputs. While this can be exponential in number of bits in the function, prior to our work, it was not clear how to create *any* algorithm from an approximately satisfying solution. Moreover, the general adversary bound is an SDP with dimension $|X|$ where $X$ is the function domain, so in general, solvers will already take time polynomial in $|X|$ to solve classically [48], so we expect that attaining this level of precision will only contribute polynomially to the overall runtime. Additionally, we show numerically that, at least for small functions, the error bound we require is easily attainable.

One may naturally wonder why a vector set that approximately satisfies the dual adversary constraints does not immediately yield an appropriate algorithm. The challenge is that the standard algorithm is based on a reflection about the span of a subset of those vectors. For example, consider a vector set that should ideally be $\{|0\rangle, |1\rangle, |0\rangle + |1\rangle\}$, but is instead $\{|0\rangle + 10^{-10}|2\rangle, |1\rangle, |0\rangle + |1\rangle\}$. The span of the first set is 2-dimensional, but the span of the second set is 3-dimensional, even though the two vector sets are very close by almost any metric. If the algorithm reflects over a space that is much larger than it should, it might not correctly evaluate the function on all inputs. Our techniques allow us to find appropriate reflections (or in fact unitaries that are close to reflection) so that the algorithms can proceed, even with errors like the example above. Along the way towards proving our main results, we also develop several tools that may be of independent interest, including a robust approximate spectral gap lemma and open-source code to find solutions to the general adversary dual.

## 1.1 Related Work

### Space Complexity and Compression

Reichardt observes that that the space used by the dual adversary algorithm is the log of the rank of $Z$, where $Z$ is the positive semidefinite matrix that optimizes the primal general adversary bound, and he notes that this provides a worst case $\log(n|X|)$ space complexity for $n$-bit functions with domain $X$ [44, 43]. Our exact compression result (Theorem 9) is of a similar flavor, except that we are using the "rank" of the dual. Barnum, Saks, and Szegedy use a different family of SDPs to characterize query complexity [5] (these SDPs can give improved performance in the case of small or zero error algorithms), and their algorithm again depends on the rank of the satisfying positive semidefinite matrix, but in the worst case uses $\log |X| + 1$ qubits.

The key tool we use in our compression application is the Johnson-Lindenstrauss lemma. The lemma guarantees that high-dimensional vectors randomly compressed into a lower-dimensional space approximately preserves the inner products of the vectors [34]. The JL lemma is used in a variety of classical applications including compressed sensing, dimensionality reduction, and machine learning [23, 49, 13], and it works even with sparse compression matrices [35]. In fact, our work, in which we compress the dual solution to an SDP, has similarities to work by So et al. [47], which uses JL compression to reduce the rank of the matrix that is the primal solution to a semidefinite programming problem, at the cost of only approximately satisfying the constraints. It is also known that the compressed dimension given by the Johnson-Lindenstrauss lemma is optimal up to constant factors [39].

The idea of relaxing SDP constraints in order to improve the space used by an algorithm has also been considered in the classical regime. Ding et al. create a storage-optimal SDP solver by relaxing constraints [22].

In the quantum arena, a natural application of the JL lemma would be to compress the size of quantum states, but Harrow et al. found that there is no such mapping that significantly reduces the dimension of quantum states while preserving the Schatten 2-norm distance with high probability [27]. However, the JL lemma was used to compress the space used by a quantum finger printing protocol [25].

Span programs (which are equivalent to the dual adversary [44, 46]) were in fact originally formulated in order to understand classical space complexity [36], and Jeffery shows lower bounds on the space complexity of function evaluation that depend on minimum span program and approximate span program sizes [31].

#### Numerical Solutions to the Dual Adversary

The idea of using classical computers to design quantum algorithms is not new. The variational quantum eigensolver iteratively uses a classical computer to make a ground state ansatz, which is then tested by a quantum computer [42]. A classical machine learning algorithm can be used to guide quantum algorithm design [4]. However, the dual adversary semidefinite programming problem is different in that it automatically produces a query-optimal algorithm, rather than an iterative process guided by classical, heuristic optimization methods.

### 1.2 Open Questions

Our techniques for space compression preserve the quantum query complexity of the original algorithm, while attempting to reduce space complexity. It would be very interesting if they could be modified to reduce space at the cost of increased query complexity; this might provide insight into one of Aaronson's 2021 open query complexity problems [1]: better understanding space and query trade offs, specifically for the problems of collision and element distinctness.

While we analyze dual adversary algorithms, these are closely related to span program algorithms. It should be possible to translate the bounds and conditions we find on the robustness of the general adversary dual into analogous bounds and conditions on span program algorithms. We are especially curious if these relaxed constraints could be related to Approximate Span Programs [30], which are another way of relaxing the constraints of standard span programs.

While we show conditions under which it is possible to create dual adversary algorithms, we do not prove lower bounds. It would be interesting to study whether, with more detailed analysis, or under additional natural conditions, the JL approach to space compression could be improved, or whether the analysis we present is optimal.

Generalizations of the general adversary bound characterize the problems of quantum state conversion [40] and quantum subspace conversion [10]. Perhaps our techniques could be extended to these additional regimes.

## 2 Preliminaries

A few notational conventions: we use $\||\psi\rangle\|$ do denote the $\ell_2$ norm of $|\psi\rangle$, $[n]$ to denote $\{1, 2, \ldots, n\}$, and $\delta_{i,j}$ to denote the Kronecker delta function. If $|\lambda\rangle$ is an eigenvector of $U$ with eigenvalue $e^{i\beta}$, we say the phase of $|\lambda\rangle$ is $\beta$. For any unitary $U$, let $P_\Theta(U)$ be the orthogonal projector onto the eigenvectors of $U$ with phase at most $\Theta$. That is, $P_\Theta(U)$ is the orthogonal projector onto $\text{span}\{|\lambda\rangle : U|\lambda\rangle = e^{i\beta}|\lambda\rangle \text{ with } |\beta| \leq \Theta\}$.

We consider the quantum query complexity and space complexity of evaluating a function $f : X \to \{0,1\}$ where $X \subseteq \{0,1\}^n$. For such a function $f$, we define $f^{-1}(b) = \{x \in X : f(x) = b\}$. For some $x \in X \subseteq \{0,1\}^n$, we are given access to an oracle $O_x$ that acts on $\mathbb{C}^n \otimes \mathbb{C}^2$ as $O_x|i\rangle|b\rangle = |i\rangle|b \oplus x_i\rangle$, where $|i\rangle$ for $i \in [n]$ and $|b\rangle$ for $b \in \{0,1\}$ are standard basis states, and $x_i$ is the $i^{\text{th}}$ bit of $x$. Given $O_x$, we would like to determine $f(x)$. We do this by implementing a bounded-error quantum query algorithm, which without loss of generality takes the form

$$U_T O_x U_{T-1} O_x \cdots U_1 O_x U_0 |\hat{0}\rangle, \tag{1}$$

followed by a two-outcome measurement that determines the output of the algorithm, where $U_0, \ldots U_T$ are unitary operations acting on a Hilbert space of size $S$, such that for every input $x \in X$, the probability of outputting $f(x)$ is at least $2/3$. The algorithm uses $T$ applications of the oracle and $\log S$ qubits of space. The bounded-error query complexity of $f$ is the minimum query complexity of any bounded-error query algorithm for $f$.

The general adversary dual is used in designing query-optimal quantum algorithms for function evaluation:

▶ **Definition 1** (General Adversary Dual). *Let $f : X \to \{0,1\}$ for $X \subseteq \{0,1\}^n$. The following semidefinite optimization problem is called the dual of the general adversary bound, or what we call the general adversary dual:*

$$\min_{\substack{m \in \mathbb{N} \\ |v_{x,j}\rangle \in \mathbb{C}^m}} \left\{ \max_{x \in X} \sum_j \||v_{x,j}\rangle\|^2 \right\} \tag{2}$$

$$s.t.\ \forall x, y \in X : f(x) \neq f(y), \qquad 1 = \sum_{j : x_j \neq y_j} \langle v_{x,j} | v_{y,j} \rangle. \tag{3}$$

While Definition 1 seeks to minimize the dimension $m$ of the vectors $\{|v_{x,j}\rangle\}_{x \in X, j \in [n]}$ (we will drop the set-building subscript and use $\{|v_{x,j}\rangle\}$ when clear from context) that also minimizes $\max_{x \in X} \sum_j \||v_{x,j}\rangle\|^2$, we note that to design an algorithm, we only need a vector set $\{|v_{x,j}\rangle\}$ that satisfies the constraints in Equation (3). This motivates the following definition, similar to converting vector sets in [3].

▶ **Definition 2** (Deciding Vector Set and Related Terms). *Let $f : X \to \{0,1\}$ for $X \subseteq \{0,1\}^n$, and let $m \in \mathbb{N}$. Then $\{|v_{x,j}\rangle \in \mathbb{C}^m\}_{x \in X, j \in [n]}$ is an $f$-deciding vector set if*

$$\forall x, y \in X : f(x) \neq f(y), \qquad 1 = \sum_{j : x_j \neq y_j} \langle v_{x,j} | v_{y,j} \rangle. \tag{4}$$

*We say the* size *of $\{|v_{x,j}\rangle\}$ is $\max_{x \in X} \sum_j \||v_{x,j}\rangle\|^2$, the* dimension *is $m$, and the* maximum rank *is* $\max_{j \in [n]} \quad \text{rank}\{|v_{x,j}\rangle : f(x) = 1\}$.

Given an $f$-deciding vector set, one can design a query algorithm to decide $f$:

▶ **Theorem 3** ([44, 40]). *For $f : X \to \{0,1\}$ with $X \subseteq \{0,1\}^n$ let $\{|v_{x,j}\rangle\}_{x \in X, j \in [n]}$ be an $f$-deciding vector set with size $A$ and dimension $m$. Then there is a bounded-error quantum query algorithm that decides $f$ with query complexity $O(A)$ and space complexity $O(\log(nm))$.*

Because any $n$-bit function can be decided in $n$ queries, we assume $A = O(n)$. Additionally, applying Cauchy-Schwarz to Equation (4), we have $A \geq 1$.

We sketch the proof of Theorem 3 to make it easier to compare with our algorithms. (For a more detailed description using similar notation, see Ref. [14, Chapter 23.6]). The subroutine used in both Theorem 3 and in our algorithms is (parallelized) phase estimation.

▶ **Lemma 4** (Phase Estimation [37, 16, 41]). *Let $U$ be a unitary that acts on $n$ qubits, and let $\delta, \Theta > 0$. Then there is a phase estimation style circuit that has precision $\Theta$, error $\delta$, acts on $n + b$ qubits for $b = O\left(\log \frac{1}{\Theta} \log \frac{1}{\delta}\right)$, and uses $O\left(\frac{1}{\Theta} \log \frac{1}{\delta}\right)$ calls to control-$U$ applied to a single instance of the state $|\psi\rangle$, such that $p_0$, the probability of outcome 0, satisfies*

$$\|P_{\Theta/2}(U)|\psi\rangle\|^2(1-\delta) - \delta \leq p_0 \leq \|P_{\Theta}(U)|\psi\rangle\|^2 + \delta. \tag{5}$$

We prove Lemma 4 in the full version of the paper [17], which relies heavily on prior analyses of phase estimation circuits.

The algorithm of Theorem 3 applies phase estimation with precision $O(1/A)$ on a unitary $U$ with an initial state $|\hat{0}\rangle$. $U$ acts on the space $\mathcal{H} = \mathbb{C} \oplus \mathbb{C}^n \otimes \mathbb{C}^m \otimes \mathbb{C}^2$, and is a product of two reflections: $U = (2\Pi_x - I)(2\Delta - I)$, where $\Pi_x = |\hat{0}\rangle\langle\hat{0}| + \sum_{i\in[n]} |i\rangle\langle i| \otimes I \otimes |x_i\rangle\langle x_i|$ (here $I$ acts on $\mathbb{C}^m$, and $|\hat{0}\rangle$ is orthogonal to $\sum_{i\in[n]} |i\rangle\langle i| \otimes I \otimes (|0\rangle\langle 0| + |1\rangle\langle 1|)$) and $\Delta$ is the orthogonal projector onto the following set of normalized vectors:

$$|\psi_y\rangle = \frac{1}{\sqrt{\nu_y}} \left( |\hat{0}\rangle + \frac{1}{\sqrt{cA}} \sum_{i\in[n]} |i\rangle|v_{y,i}\rangle|y_i\rangle \right) \qquad \forall y : f(y) = 1, \tag{6}$$

$$\text{s.t. } \nu_y = 1 + \frac{1}{cA} \sum_{i\in[n]} \||v_{y,i}\rangle\|^2 \leq 1 + 1/c, \tag{7}$$

where $\nu_y \geq 1$ is chosen to normalize $|\psi_y\rangle$, and $c$ is a constant chosen depending on the desired success probability. We note $(2\Pi_x - I)$ requires 2 uses of $O_x$ to implement, and $(2\Delta - I)$ depends on the choice of the deciding vector set but is independent of the input $x$.

Then when $f(x) = 1$ and $c = 2$, $|\hat{0}\rangle$ has high overlap with $|\psi_x\rangle$, which is easily verified to be a 0-phase eigenvector of $U$, so by Lemma 4, the probability of measuring a phase of 0 when we perform phase estimation is large when $\delta$ (the error of phase estimation) is a small constant.

On the other hand, when $f(x) = 0$, we consider the following normalized vector:

$$|\phi_x\rangle = \frac{1}{\sqrt{\mu_y}} \left( |\hat{0}\rangle - \sqrt{cA} \sum_{i\in[n]} |i\rangle|v_{x,i}\rangle|\bar{x}_i\rangle \right) \tag{8}$$

$$\text{s.t. } \mu_x = 1 + cA \sum_{i\in[n]} \||v_{x,i}\rangle\|^2 \leq 1 + cA^2, \tag{9}$$

where $\mu_y \geq 1$ is chosen to normalize the vector. Because $\{|v_{y,i}\rangle\}$ is a deciding vector set, we have $\forall y : f(y) = 1, \langle\psi_y|\phi_x\rangle = 0$. Thus $\{|\phi_x\rangle\}$ is orthogonal to $\Delta$. Next we use the effective spectral gap lemma:

▶ **Lemma 5** (Effective Spectral Gap Lemma [40]). *Let $\Pi, \Delta$ be orthogonal projectors, let $U = (2\Pi - I)(2\Delta - I)$, and $\Delta|w\rangle = 0$. Then*

$$\|P_{\Theta}(U)\Pi|w\rangle\| \leq \Theta/2\||w\rangle\|. \tag{10}$$

Then the probability of measuring a phase of 0 when we perform phase estimation when $f(x) = 0$, by Lemma 4 is upper bounded by a term that depends on $\|P_{\Theta}(U)|\hat{0}\rangle\|^2 = \mu_y\|P_{\Theta}(U)\Pi_x|\phi_x\rangle\|^2$. Applying Lemma 5, and using the fact that $\mu_y = O(A^2)$, we see this is small when $\Theta$, the precision of phase estimation, is chosen to be $O(1/A)$. Since $\delta$ (the error of phase estimation) is chosen to be a small constant, by Lemma 4 this leads to a bounded-error algorithm with query complexity $O(A)$, and space complexity $\log(1 + 2nm) + O(\log A) = O(\log(nm))$, as claimed in Theorem 3.

## 3 Robust Dual Adversary Algorithm

In this section, we show that the dual adversary algorithm has robustness, in that it tolerates errors and flexibility in how it is defined. As described in Section 2, we want to create a bounded-error quantum query algorithm for a Boolean function $f : X \to \{0,1\}$, for $X \subseteq \{0,1\}^n$. Similar to the standard algorithm, our robust algorithm will involve applying phase estimation to a unitary $U$ that acts on the space $\mathcal{H} = \mathbb{C} \oplus \mathbb{C}^n \otimes \mathbb{C}^m \otimes \mathbb{C}^2$. We perform phase estimation on $U$ with a state $|\hat{0}\rangle \in \mathcal{H}$.

We now describe $U$. As in Section 2, we define the orthogonal projector $\Pi_x = |\hat{0}\rangle\langle\hat{0}| + \sum_{i\in[n]} |i\rangle\langle i| \otimes I \otimes |x_i\rangle\langle x_i|$ on $\mathcal{H}$ where $I$ acts on $\mathbb{C}^m$, and $|\hat{0}\rangle$ is orthogonal to $\sum_{i\in[n]} |i\rangle\langle i| \otimes I \otimes (|0\rangle\langle 0| + |1\rangle\langle 1|)$. Notice that $\Pi_x$ can be implemented with two applications of the oracle $O_x$. Let $R$ be another unitary that acts on the same space as $\Pi_x$, but $R$ need not be a reflection. Let $U = (2\Pi_x - I)R$.

▶ **Theorem 6.** *Let $\delta, \nu_x, \mu_x > 0$, $\varepsilon_\psi, \varepsilon_\phi \geq 0$, and let $U = (2\Pi_x - I)R$ as defined above, so $U$ acts on $O(\log nm)$ qubits. Consider $0 < \theta \leq 1$. Suppose there are sets of (not necessarily normalized) vectors $\{|\psi_x\rangle = \frac{1}{\sqrt{\nu_x}}(|\hat{0}\rangle + |\eta_x\rangle)\}_{x:f(x)=1}$ and $\{|\phi_x\rangle = \frac{1}{\sqrt{\mu_x}}(|\hat{0}\rangle + |\eta_x\rangle)\}_{x:f(x)=0}$ where $\forall x \in X, \langle\eta_x|\hat{0}\rangle = 0$, and furthermore, that*

1. *$\forall x : f(x) = 1, \quad \|(I-U)|\psi_x\rangle\| \leq \varepsilon_\psi$ and*
2. *$\forall x : f(x) = 0, \quad \Pi_x|\eta_x\rangle = 0$ and $\|(I+R)|\phi_x\rangle\| \leq \varepsilon_\phi$.*

*Then the probability of measuring a phase of 0 if we do phase estimation on $U$ with initial state $|\hat{0}\rangle$ with precision $\theta$ and error $\delta$ when $f(x) = 1$ is at least*

$$\left(\sqrt{\nu_x(1 - \frac{5\varepsilon_\psi^2}{\theta^2})} - \sqrt{\nu_x \||\psi_x\rangle\|^2 - 1}\right)^2 (1-\delta) - \delta, \tag{11}$$

*and when $f(x) = 0$ is at most*

$$\mu_x \left(\varepsilon_\phi/2 + \theta/2 \||\phi_x\rangle\|\right)^2 + \delta. \tag{12}$$

*This algorithm uses $O\left(\frac{1}{\theta}\log\frac{1}{\delta}\right)$ queries and $O\left(\log(nm) + \log\frac{1}{\theta}\log\frac{1}{\delta}\right)$ qubits.*

Theorem 6 extends the robustness of the algorithm used to prove Theorem 3 (as described in Section 2) in several ways. In the standard analysis, $\varepsilon_\psi$ and $\varepsilon_\phi$ are both 0, whereas we now allow them to be non-zero. In addition, Theorem 6 allows for imperfect alignment between the vector sets $\{|\psi_x\rangle\}$ and $\{|\phi_x\rangle\}$ and the unitary $U$. This will be the key for our applications in the following sections. Additionally, in the standard algorithm, $R$ is chosen to be a reflection, but in Theorem 6, $R$ can be any unitary that satisfies the criterion of Theorem 6. While none of the applications we describe in this paper use this flexibility in the design of $R$, it might be helpful in future use cases.

To prove Theorem 6, we need the following two lemmas, proved in the full version [17]:

▶ **Lemma 7.** *Let $U$ be a unitary and $0 < \Theta \leq 1$. If $\|(I-U)|\psi_x\rangle\|^2 \leq \varepsilon$, then $\|P_\Theta(U)|\psi_x\rangle\|^2 \geq 1 - \frac{1.1\varepsilon}{\Theta^2}$.*

Lemma 7 tells us that if a unitary $U$ approximately preserves a state $|\psi_x\rangle$, then $|\psi_x\rangle$ has high overlap with the low phase eigenspace of $U$. This gives us flexibility when $f(x) = 1$, in that our initial state need not have high overlap with the 0-phase space of $U$, but instead we only require high overlap with the low-phase-eigenspace of $U$. The proof of Lemma 7 proceeds by decomposing $|\psi_x\rangle$ into its eigenbasis with respect to $U$, and showing that $\epsilon$ serves to bound the amount of amplitude $|\psi_x\rangle$ can have in states with eigenvalues larger than $\Theta$.

▶ **Lemma 8** (Robust Approximate Spectral Gap Lemma). *Let $\varepsilon \geq 0$, $\Pi$ be an orthogonal projector, $R$ be a unitary, and $U = (2\Pi - I)R$. For $\Theta > 0$, if $\|(I+R)|w\rangle\| \leq \varepsilon$, then*

$$\|P_\Theta(U)\Pi|w\rangle\| \leq \frac{\varepsilon}{2} + \frac{\Theta}{2}\||w\rangle\|. \tag{13}$$

Lemma 8 generalizes the standard approximate spectral gap lemma (Lemma 5), in which $R$ is a reflection and $\varepsilon = 0$. In particular, Lemma 8 shows that when $R$ is not a reflection and when $|w\rangle$ is not exactly given a phase of $-1$ by $R$, a variant of the approximate spectral gap lemma still holds.[1] The proof of Lemma 8 closely follows the proof approach of [40, Lemma 4.2], which does not use Jordan's Lemma, but instead directly uses a series of triangle inequalities and observations of preserved subspaces to obtain the result.

**Proof of Theorem 6.** We first consider the case of $x$ such that $f(x) = 1$. By Lemma 4, the probability that we get an outcome of 0 when we perform phase estimation on the unitary $U$ with initial state $|\hat{0}\rangle$ with precision $\Theta$ and accuracy $\delta$ is at least $\|P_{\Theta/2}(U)|\hat{0}\rangle\|^2(1-\delta) - \delta$. Now

$$\begin{aligned}
\|P_{\Theta/2}(U)|\hat{0}\rangle\| &= \|P_{\Theta/2}(U)\sqrt{\nu_x}|\psi_x\rangle - P_{\Theta/2}(U)|\eta_x\rangle\| \\
&\geq \sqrt{\nu_x}\|P_{\Theta/2}(U)|\psi_x\rangle\| - \|P_{\Theta/2}(U)|\eta_x\rangle\| \qquad \text{(reverse triangle inequality)} \\
&\geq \sqrt{\nu_x\left(1 - \frac{5\varepsilon_\psi^2}{\Theta^2}\right)} - \||\eta_x\rangle\|, \qquad (14)
\end{aligned}$$

where the first term in the final line combines Lemma 7 and the assumption that $\|(I - U)|\psi_x\rangle\| \leq \varepsilon_\psi$, so $\|(I-U)|\psi_x\rangle\|^2 \leq \varepsilon_\psi^2$, and the second term uses the fact that projectors can only decrease the $\ell_2$ norm of a vector. Thus, using that $\||\eta_x\rangle\| = \sqrt{\nu_x\||\psi_x\rangle\|^2 - 1}$, we have

$$\|P_{\Theta/2}(U)|\hat{0}\rangle\|^2(1-\delta) - \delta \geq \left(\sqrt{\nu_x(1 - \frac{5\varepsilon_\psi^2}{\Theta^2})} - \sqrt{\nu_x\||\psi_x\rangle\|^2 - 1}\right)^2 (1-\delta) - \delta. \tag{15}$$

When $f(x) = 0$, by Lemma 4, the probability that we get an outcome of 0 when we perform phase estimation on $U$ with initial state $|\hat{0}\rangle$ with precision $\Theta$ and accuracy $\delta$ is at most

$$\|P_\Theta(U)|\hat{0}\rangle\|^2 + \delta = \mu_x\|P_\Theta(U)\Pi_x|\phi_x\rangle\|^2 + \delta, \tag{16}$$

since by assumption, $\Pi_x|\phi_x\rangle = 1/\sqrt{\mu_x}|\hat{0}\rangle$. Then from Lemma 8 and our assumption that $\|(I+R)|\phi_x\rangle\| \leq \varepsilon_\phi$, we have

$$\|P_\Theta(U)\Pi_x|\phi_x\rangle\| \leq \varepsilon_\phi/2 + \Theta/2\||\phi_x\rangle\|. \tag{17}$$

Combining Equation (16) and Equation (17) gives us a probability of outcome 0 of at most

$$\mu_x\left(\varepsilon_\phi/2 + \Theta/2\||\phi_x\rangle\|\right)^2 + \delta. \tag{18}$$

Finally the query complexity and space complexity come from the requirements of phase estimation Lemma 4, and that $2\Pi_x - I$ can be implemented with two uses of the oracle. ◀

[1] We note Lemma 8 is similar to Lemma 3.4 in [33], except we allow $R$ to not be a reflection.

## 4 Compressing the Dual Adversary Algorithm

In this section, we consider how and when it is possible to reduce the space complexity of the quantum algorithm built from the general adversary dual. In particular, our goal is to take an $f$-deciding vector set, reduce its dimension and hence create an algorithm which requires fewer qubits to implement.

Our first result, Theorem 9, is a simple compression scheme that shows that an $f$-deciding vector set on an $n$-bit function with maximum rank $\kappa'$ and size $A$ can be compressed to an $f$-deciding vector set with dimension $\kappa'$ and size at most $A$ (see Definition 2 for terminology). The number of qubits required by the resulting algorithm is then $O(\log(n\kappa'))$ and the query complexity is $O(A)$, by Theorem 3 and using that $A = O(n)$. Notice that $\kappa'$ is at most the number of 1-valued inputs, but if many of the $f$-deciding vectors are linearly dependent, the maximum rank could be much smaller.

We note that an $f$-deciding vector set is also an $\neg f$-deciding vector set by Definition 2, where $\neg f$ is the negation of $f$. Also, a $\neg f$-deciding vector set can be used to design a bounded error quantum algorithm for deciding $f$ by negating the output of the algorithm. Thus for a given deciding vector set, we can minimize the space used by the algorithm by considering either $\neg f$ or $f$.

▶ **Theorem 9.** *Given an $f$-deciding vector set with maximum rank $\kappa'$ and size $A$, we can construct an $f$-deciding vector set with dimension $\kappa'$ and size at most $A$, resulting in an algorithm that decides $f$ with query complexity $O(A)$ and space complexity $O(\log(n\kappa'))$.*

To prove Theorem 9, we apply a series of unitaries to rotate the vectors of the deciding vector set into a smaller dimensional space. This is possible because if we apply the same unitary to two vectors, their inner product is preserved. Our full proof appears in Ref. [17].

The next natural question is whether we can *approximate* the solution to the general adversary dual in a lower dimension. We answer this question by considering a compression of the vectors in a deciding vector set with guarantees from the Johnson-Lindenstrauss lemma, which approximately preserve the inner products of the vectors.

It turns out that this straightforward idea is not trivial to implement. The first challenge is that we must preserve the tensor product structure of our vectors in order to ensure that the query algorithm can apply queries, so we must be careful about the part of the vectors that we compress. The second challenge is that the compression only approximately preserves the inner products of the compressed vectors so we need the robust dual adversary algorithm described in Theorem 6.

Formally stated in Theorem 14, given an $f$-deciding vector set with maximum rank $\kappa'$ and size $A$, we show how to build a quantum algorithm that succeeds with probability $2/3$ and operates in a Hilbert space of dimension $O((\kappa'^2 + A^4\kappa')n)$ with quantum query complexity $O(A)$. Since $A = O(n)$, as discussed below Theorem 3, the number of qubits needed to run the algorithm is no more than $O(\log(\kappa' n))$.

Thus, this approximate compression using the JL lemma achieves the same space complexity as the exact compression of Theorem 9, to within a constant multiplicative factor. This may seem surprising that we are not able to do better, since we are no longer requiring the constraints are exactly satisfied. However, the compression dimension in the JL lemma has a polynomial dependence on the allowed error, and since the amount of error we can tolerate roughly scales with maximum rank, we do not get as much compression as one might hope for.

We now describe at a high level how we prove Theorem 14. While an optimal deciding vector set might use complex vectors, the following lemma, which we prove in Ref. [17], shows that at a small cost in increased vector dimension, we can restrict to real vectors:

▶ **Lemma 10.** *If there is an $f$-deciding vector set with complex numbers of dimension $m$ and size $A$, there exists an $f$-deciding vector set with only real numbers of dimension $2m$ and size $A$.*

The proof of Lemma 10 proceeds by creating a new vector set from the original where each new real vector consists of the real part of the original complex vector stacked on top of the imaginary part of the original complex vector.

While the standard Johnson-Lindenstrauss lemma guarantees that there is a compression matrix that approximately preserves the $l_2$-norm difference of any two vectors in a set, we use the following corollary, which shows that the compression approximately preserves inner products in addition to distances, which we prove in Ref. [17]. Our proof of Corollary 11 is similar to a similar result in [38], except that we do not assume the vectors have norm 1.

▶ **Corollary 11.** *Given $\varepsilon > 0$, a set of finite vectors $V \subset \mathbb{R}^d$, and a number $N > 8\ln(|V|)/\varepsilon^2$, there is a compression matrix $S \in \mathbb{R}^{N\times d}$ such that for $|v\rangle, |u\rangle \in V$,*

$$(S|u\rangle)^\dagger(S|v\rangle) = \langle u|v\rangle \pm 2\varepsilon(\||u\rangle\|^2 + \||v\rangle\|^2). \tag{19}$$

The Johnson-Lindenstrauss lemma guarantees the existence of such a compression matrix $S$, and it can be found probabilistically by sampling random projections. It requires $O(|X|n)$ time to find such a satisfying projection via random sampling [18]. For our purposes, this contributes to classical preprocessing time and space resources, and not towards the quantum query complexity or quantum space use of the quantum algorithm itself. In particular, this sampling requires no queries, so does not contribute to the query complexity.

We now describe how we use Johnson-Lindenstrauss to compress our deciding vector set. Let $\{|v_{x,i}\rangle\}$ be a real $f$-deciding vector set with size $A$, dimension $m$, and maximum rank $\kappa'$. Define $\{|\psi_x\rangle\}_{x:f(x)=1}$ and $\{|\phi_x\rangle\}_{x:f(x)=0}$ as in Equations (6) and (8) in Section 2. Let $\kappa$ be the rank of $\{|\psi_x\rangle\}_{x:f(x)=1}$. (We show in Ref. [17] that $\kappa' \leq \kappa \leq 2n\kappa'$.)

To compress our vectors, it suffices to compress their orthonormal basis. Let $\{|\zeta_j\rangle\}_{j\in[\kappa]}$ be an orthonormal basis for the space spanned by $\{|\psi_x\rangle\}_{x:f(x)=1}$. Then there are (non-unique) real numbers $\{\alpha_{j,x} \in \mathbb{R}\}_{j\in[n],x\in f^{-1}(1)}$ such that

$$|\zeta_j\rangle = \sum_{x:f(x)=1} \alpha_{j,x}|\psi_x\rangle. \tag{20}$$

We will approximately preserve the structure of the vectors $|\zeta_j\rangle$ in the compression. Thus we define their components

$$|v_{j,i,b}\rangle = \sum_{\substack{x:f(x)=1\\ x_i=b}} \frac{\alpha_{j,x}}{\sqrt{\nu}_x}|v_{x,i}\rangle, \quad \forall j \in [\kappa], i \in [n], b \in \{0,1\}. \tag{21}$$

We will use the random compression matrix $S$ from Corollary 11 to compress the following set of vectors to error $\varepsilon$, as in Corollary 11, (and the compression dimension $N$ will be chosen later to achieve the desired value of $\varepsilon$):

$$\{|v_{j,i,b}\rangle\}_{j\in[\kappa],i\in[n],b\in\{0,1\}} \bigcup \{|v_{y,i}\rangle\}_{y:f(y)=0,i\in[n]}. \tag{22}$$

We use these compressed vectors to define

$$\begin{aligned}
\forall x : f(x) = 1, \quad |\psi'_x\rangle &= \left[|\hat{0}\rangle\langle\hat{0}| + (I \otimes S \otimes I)\right]|\psi_x\rangle,\\
\forall x : f(x) = 0, \quad |\phi'_x\rangle &= \left[|\hat{0}\rangle\langle\hat{0}| + (I \otimes S \otimes I)\right]|\phi_x\rangle,\\
\forall j \in [\kappa], \quad |\zeta'_j\rangle &= \left[|\hat{0}\rangle\langle\hat{0}| + (I \otimes S \otimes I)\right]|\zeta_j\rangle.
\end{aligned} \tag{23}$$

As one would expect, the primed, compressed versions of these vectors have approximately the properties of the uncompressed version, as follows:

▶ **Lemma 12.** *For* $\{|\zeta'_j\rangle\}_{j\in[\kappa]}$, $\{|\psi'_x\rangle\}_{x:f(x)=1}$ *and* $\{|\phi'_x\rangle\}_{x:f(x)=0}$ *as described in Equation* (23)*, these vectors have the following properties*

1. $\forall j,l \in [\kappa], \langle\zeta'_j|\zeta'_l\rangle \in \delta_{j,l} \pm 4\varepsilon$
2. $\forall j \in [\kappa], x \in f^{-1}(0), |\langle\zeta'_j|\phi'_x\rangle| \leq 2\varepsilon(c+1)A.$
3. $\forall x : f(x) = 1, \left|\||\psi'_x\rangle\|^2 - 1\right| \leq 4\varepsilon\kappa$ *and* $\forall x : f(x) = 0, |\||\phi'_x\rangle\| - 1| \leq 3\varepsilon.$

The proof of Lemma 12 uses Corollary 11 in fairly straightforward ways.

Let $\Delta'$ be the orthogonal projector onto the space spanned by $\{|\psi'_x\rangle\}_{x:f(x)=1}$ and the reflection $R$ be $2\Delta' - I$. By definition, observe that $R|\psi'_x\rangle = |\psi'_x\rangle$, so Theorem 6 Item 1 is satisfied with $\varepsilon_\psi = 0$. Additionally, because $|\phi'_x\rangle$ has the structure

$$|\phi'_x\rangle \propto |\hat{0}\rangle + \sum_{i\in[n]} |i\rangle|v'_{x,i}\rangle|\bar{x}_i\rangle \tag{24}$$

we have $\Pi_x|\phi'_x\rangle \propto |\hat{0}\rangle$, as required by Theorem 6 Item 2. All that is left is to show that $\|(I+R)|\phi'_x\rangle\| \leq \varepsilon_\phi$.

▶ **Lemma 13.** *Consider* $\varepsilon$ *so that* $\varepsilon\kappa < 1/12$*. For* $R$ *as defined below Lemma 12 and* $|\phi'_x\rangle$ *defined using Equations* (8) *and* (23)*, we have* $\|(I+R)|\phi'_x\rangle\| \leq 8\varepsilon(c+1)A\sqrt{\kappa}$*.*

We prove Lemma 13 in Ref. [17]. The main idea is to write $\Delta'$ in terms of the vectors $\{|\zeta'\rangle\}$, and then use Lemma 12 Item 2. Along the way, we show how to use Gram-Schmidt to build an orthonormal basis from $\{|\zeta'_j\rangle\}$, which is already almost orthonormal by Lemma 12 Item 1 (see Ref. [17] for details).

With Lemma 13 in hand, the conditions of Theorem 6 are satisfied and we apply it to our compressed vectors.

▶ **Theorem 14.** *Consider a Boolean function* $f : X \to \{0,1\}$ *where* $X \subseteq \{0,1\}^n$ *and an* $f$*-deciding vector set with maximum rank* $\kappa$ *and size* $A$*. Using Johnson-Lindenstrauss compression, we can compress the* $f$*-deciding vector set to produce a quantum algorithm that correctly evaluates* $f(x)$ *with probability* $2/3$ *for every input* $x \in X$ *with* $O(A)$ *quantum query complexity. The algorithm uses* $O(\log \kappa n)$ *qubits.*

**Proof of Theorem 14.** We set

$$\Theta = \frac{1}{4\sqrt{c}A}, \quad \varepsilon = \min\left\{\frac{1}{4c\kappa}, \frac{1}{32\sqrt{c}(c+1)A^2\sqrt{\kappa}}\right\}, \quad \delta = \frac{1}{25}, \quad \text{and} \quad c = 100. \tag{25}$$

With these choices, we bound the failure probability below $1/3$ for all inputs $x$. (Note $c$ appears in Equations (6) and (8).)

**Case $f(x) = 1$.** By Theorem 6, for $x$ such that $f(x) = 1$, we have that when $\varepsilon_\psi = 0$, the probability of measuring a phase of 0 is at least

$$\left(\sqrt{\nu_x} - \sqrt{\nu_x\||\psi'_x\rangle\|^2 - 1}\right)^2 (1-\delta) - \delta. \tag{26}$$

We know that $1 \leq \nu_x \leq 1 + 1/c = 1.01$ from Equation (7) and $|\||\psi'_x\rangle\|^2 - 1| \leq 4\varepsilon\kappa \leq 1/100$ from Lemma 12 Item 3 and Equation (25) when $c = 100$, so Equation (26) is at least

$$\left(\sqrt{1} - \sqrt{1.01 \cdot 1.01 - 1}\right)^2 (1-\delta) - \delta \geq 2/3 \tag{27}$$

where the final inequality is satisfied for $\delta = 1/25$.

**Case $\mathbf{f(x) = 0}$.** By Theorem 6, for $x$ such that $f(x) = 0$, we have that the probability of measuring a phase of 0 is at most

$$\mu_x \left( \varepsilon_\phi + \frac{\Theta}{2} \| |\phi'_x\rangle \| \right)^2 + \delta. \tag{28}$$

We know that $\mu_x \leq 1 + cA^2$ from Equation (9), $\varepsilon_\phi \leq 8\varepsilon(c+1)A\sqrt{\kappa}$ from Lemma 13, and $\| |\phi'_x\rangle \| \leq 1 + 3\varepsilon$ from Lemma 12 Item 3. So the probability of measuring a 0 phase is at most

$$(1 + cA^2) \left( 8\varepsilon(c+1)A\sqrt{\kappa} + \frac{\Theta}{2}(1 + 3\varepsilon) \right)^2 + \delta. \tag{29}$$

With $\Theta$, $\varepsilon$, $\delta$, and $c$ as set in Equation (25), continuing from Equation (29), we have that probability of failure when $f(x) = 0$ is at most

$$(1 + cA^2) \left( \frac{1}{4\sqrt{c}A} + \frac{1}{4\sqrt{c}A} \right)^2 + \delta \leq \frac{1}{4cA^2} + \frac{1}{4} + \delta \leq \frac{1}{3} \tag{30}$$

since $A \geq 1$.

To achieve this compression with $\varepsilon$ as desired we look to Corollary 11 to see what compression dimension is achievable. From Equation (22), we see we are compressing at most $3|X|n$ vectors. Thus we require a compression dimension

$$N = O(\log(|X|n)/\varepsilon^2) = O((\kappa^2 + A^4\kappa)\log(|X|n)). \tag{31}$$

Then since our unitary $U$ acts on $O(\log(nN))$ qubits, by Theorem 6, the space complexity is

$$O(\log nN + \log A) = O\left(\log\left(An(\kappa^2 + A^4\kappa)\log(|X|n)\right)\right) = O(\log(\kappa n)). \tag{32}$$

where we've used that $|X| \leq 2^n$ and $A = O(n)$. Also, from Theorem 6, the query complexity is $O(A)$. ◀

▶ **Corollary 15.** *If there are polynomially many 1-valued or 0-valued inputs to a function $f : X \to \{0, 1\}$, then there is a query-optimal quantum algorithm that evaluates $f$, and that uses $O(\log n)$ qubits.*

**Proof.** Notice that $\kappa \leq n_1$ where $n_1$ the number of 1-valued inputs to $f$. When $\kappa = O(n^d)$ for $d = O(1)$, by Theorem 9 or Theorem 14, we have that for any deciding vector set for $f$ with size $A$, we can create an algorithm with query complexity $O(A)$ whose space complexity is $O(\log n)$. Since there is always a deciding vector set for $f$ with size $A$ such that $O(A)$ is the optimal query complexity of $f$ [44], our results imply that there exists a query-optimal algorithm that uses logarithmic space. For the case of polynomially many 0-valued inputs, we use $\neg f$. ◀

## 5 Algorithm from a Numerical Solution

While the dual adversary provides a method of designing optimal query algorithms, in general it might be hard to find an optimal solution. However, since the problem can be formulated as a semidefinite program, we can find a numerical solution. We show that numerical solutions that only approximately satisfy the dual adversary constraints can be used to produce bounded-error quantum algorithms within a constant factor of the objective function value of the numerical solution.

We first provide a theoretical result that shows how an algorithm can be constructed from an *approximate* deciding vector set, a vector set that only approximately satisfies Equation (3). Then we introduce a classical Python package implemented specifically to solve the general adversary dual SDP and show that its solutions often satisfy the error bounds required by our theoretical results.

## 5.1 Application of Robust Dual Adversary Algorithm

In previous sections we assumed access to an exact solution to the general adversary dual which we used to produce approximate solutions. In this section, we use a finite precision SDP solver to obtain an approximate solution and then build a bounded-error quantum algorithm with the robust dual adversary algorithm described in Theorem 6.

▶ **Theorem 16.** *Consider a Boolean function $f : X \to \{0,1\}$ where $X \subseteq \{0,1\}^n$ and an approximate $f$-deciding vector set $\{|v_{x,j}\rangle\}_{x\in X, j\in[n]}$ in the sense that*

$$\varepsilon \coloneqq \max_{x,y:f(x)\neq f(y)} |\langle\psi_x|\phi_y\rangle|, \tag{33}$$

*is small as defined below, where $|\psi_x\rangle$, $|\phi_x\rangle$ are as in Equations (6) and (8). Let $A$ be the size and $m$ be the dimension of the approximate $f$-deciding vector set, with size and dimension defined as in Definition 2. Consider a matrix $M$ with rows $|\psi_x\rangle_{x:f(x)=1}$. Let the singular values of $M$ be $s_1 \geq s_2 \geq \cdots \geq s_\kappa \geq s_{\kappa+1} \coloneqq 0$ and $n_1$ be $|\{x : f(x) = 1\}|$. If there exists $\kappa^* \in [\kappa]$ such that*

$$\varepsilon \leq \frac{1}{\sqrt{n_1}}\left(\frac{s_{\kappa^*}}{2\sqrt{c}A} - s_{\kappa^*+1}\right) \quad \text{and} \quad s_{\kappa^*+1} \leq \frac{1}{2\sqrt{1000c}A}, \tag{34}$$

*then there is a quantum algorithm that correctly evaluates $f$ with probability at least $2/3$ with at most $O(A)$ queries and $O(\log(nm))$ qubits.*

Given any numerical solution, we can simply set $\kappa^* = \kappa$ in which case we require $\varepsilon < s_\kappa/(2\sqrt{cn_1}A)$. However, if the singular values fall off sharply then we can obtain a less stringent constraint on $\varepsilon$, which in turn leads to less precision required by the numerical solver. So in practice, we search for any $\kappa^* \in [\kappa]$ that gives a large enough bound to accommodate the $\varepsilon$ we observe in our numerical solution.

To prove Theorem 16, we apply Theorem 6. For this application, we set $R$ to be equal to a reflection over the space spanned by the first $\kappa^*$ right singular vectors of $M$, the matrix whose rows are the vectors $\{|\psi_x\rangle\}_{x:f(x)=1}$. We use the following two lemmas to show that this reflection approximately preserves the vectors $\{|\psi_x\rangle\}_{x:f(x)=1}$ and mostly destroys vectors that are almost orthogonal to all vectors in $\{|\psi_x\rangle\}_{x:f(x)=1}$. Their proofs, found in Ref. [17], use standard results from the singular value decomposition approach to approximating of matrices.

▶ **Lemma 17.** *Let $M$ be the matrix whose rows are the vectors $\{|\psi_x\rangle\}_{x\in X_1}$ for some set $X_1$, and denote $M$'s singular values by $s_1 \geq s_2 \geq \cdots \geq s_\kappa$. Let $\Delta'$ be the orthogonal projector onto the first $\kappa^*$ right singular vectors of $M$. Then $\forall x \in X_1$, $(2\Delta' - I)|\psi_x\rangle = |\psi_x\rangle + |\eta\rangle$, where $\||\eta\rangle\| \leq 2s_{\kappa^*+1}$ and $s_{\kappa^*+1} = 0$ if $\kappa^* = \kappa$.*

▶ **Lemma 18.** *For disjoint sets $X_1$ and $Y$ and sets of vectors $\{|\psi_x\rangle\}_{x\in X_1}$ and $\{|\phi_y\rangle\}_{y\in Y}$ such that*

$$|\langle\psi_x|\phi_y\rangle| \leq \varepsilon \quad \forall x \in X_1, y \in Y, \tag{35}$$

*let $M$ be the matrix whose rows are the vectors $\{|\psi_x\rangle\}_{x\in X_1}$ and denote its singular values by $s_1 \geq s_2 \geq \cdots \geq s_\kappa$. Let $\Delta'$ be the orthogonal projector onto the first $\kappa^*$ right singular vectors of $M$. Then $\forall y \in Y$, $(2\Delta' - I)|\phi_y\rangle = |\eta\rangle$, where $\||\eta\rangle\| \leq \frac{s_{\kappa^*+1}+\varepsilon\sqrt{|X_1|}}{s_{\kappa^*}}$ and $s_{\kappa^*+1} = 0$ if $\kappa^* = \kappa$.*

We now use Lemmas 17 and 18 to apply Theorem 6 and prove Theorem 16.

**Proof of Theorem 16.** Let $\Delta'$ be the orthogonal projector onto the first $\kappa^*$ right singular vectors of $M$, and set $R = 2\Delta' - I$ so as in Theorem 6, $U = (2\Pi_x - I)R$. Then

$$\begin{aligned}
\varepsilon_\psi &\coloneqq \|(I-U)|\psi_x\rangle\| = \||\psi_x\rangle - (2\Pi_x - I)R|\psi_x\rangle\| \\
&= \||\psi_x\rangle - (2\Pi_x - I)(|\psi_x\rangle + |\eta_x\rangle)\| \\
&= \|(2\Pi_x - I)|\eta_x\rangle\| \leq 2s_{\kappa^*+1} \quad \text{(By Lemma 17 and defs of } \Pi_x, |\psi_x\rangle\text{)} \\
\varepsilon_\phi &\coloneqq \|(I+R)|\phi_y\rangle\| = \|2\Delta'|\phi_y\rangle\| \leq 2\frac{s_{\kappa^*+1} + \varepsilon\sqrt{n_1}}{s_{\kappa^*}}. \quad \text{(By Lemma 18)}
\end{aligned}$$

Then by Theorem 6, if $f(x) = 1$, the probability of measuring a phase of 0 is at least

$$\left(\sqrt{\nu_x(1 - \frac{5\varepsilon_\psi^2}{\Theta^2})} - \sqrt{\nu_x\||\psi_x\rangle\|^2 - 1}\right)^2 (1-\delta) - \delta \tag{36}$$

$$\geq \left(\sqrt{(1 - \frac{20s_{\kappa^*+1}^2}{\Theta^2})} - \sqrt{\nu_x\||\psi_x\rangle\|^2 - 1}\right)^2 (1-\delta) - \delta \tag{37}$$

$$\geq \left(\sqrt{1 - \frac{20s_{\kappa^*+1}^2}{\Theta^2}} - \sqrt{1/c}\right)^2 (1-\delta) - \delta \geq 2/3 \tag{38}$$

where we set $c = 100$, $\delta = 1/25$, and $\Theta = (2\sqrt{c}A)^{-1}$. In addition, we used that $\nu_x$ is between 1 and $1 + 1/c$ and that $\||\psi_x\rangle\| = 1$, both by Equation (6). Recall that $s_{\kappa^*+1}^2 \leq \Theta^2/1000$ by assumption.

In the other case, where $f(x) = 0$, the probability of measuring a phase of 0 is at most

$$\mu_x\left(2\frac{s_{\kappa^*+1} + \varepsilon\sqrt{n_1}}{s_{\kappa^*}} + \Theta/2\||\phi_x\rangle\|\right)^2 + \delta \leq (1+cA^2)\left(2\frac{s_{\kappa^*+1} + \varepsilon\sqrt{n_1}}{s_{\kappa^*}} + \Theta/2\right)^2 + \delta \tag{39}$$

$$= (1+cA^2)\left(\frac{1}{4\sqrt{c}A} + \frac{1}{4\sqrt{c}A}\right)^2 + \delta \tag{40}$$

$$\leq 1/3 \qquad \text{(by the proof of Theorem 14)}$$

when

$$\frac{s_{\kappa^*+1} + \varepsilon\sqrt{n_1}}{s_{\kappa^*}} < \frac{1}{8\sqrt{c}A} \iff \varepsilon < \frac{1}{\sqrt{n_1}}\left(\frac{s_{\kappa^*}}{2\sqrt{c}A} - s_{\kappa^*+1}\right). \tag{41}$$

The algorithm uses within a multiplicative factor of $\frac{1}{\Theta}\log(\frac{1}{\delta}) = 20A\log(20)$ queries by Lemma 4. ◀

The obvious question is whether the conditions in Theorem 16 are met in practice. We show in the next subsection that the conditions are met in the vast majority of numerical solutions to the general adversary dual for the random Boolean functions we find on functions of up to 25 bits with domain size 32.

## 5.2 Experiments

In this section, we describe how the numerical error (defined in Equation (33)) behaves in practice.

The general adversary dual (Definition 1) is a semidefinite programming (SDP) problem. In order to solve the SDP numerically, we first reformulate it in the following standard form:

$$\min_{\mathbf{X}\in\mathbf{S}^n} \operatorname{tr}(\mathbf{C}\mathbf{X}) \quad \text{s.t.} \quad \mathcal{A}(\mathbf{X}) = \mathbf{b} \quad \text{and} \quad \mathbf{X} \succeq 0 \tag{42}$$

where $\mathbf{S}^n$ is the set of $n \times n$ symmetric matrices, $\mathbf{C} \in \mathbf{S}^n$, $\operatorname{tr}(\cdot)$ is the trace, $\mathbf{b} \in \mathbb{R}^m$ is the constraint vector, and

$$\mathcal{A}(\mathbf{X}) \coloneqq [\operatorname{tr}(\mathbf{A}^1\mathbf{X}), \ldots, \operatorname{tr}(\mathbf{A}^m\mathbf{X})]^\top \tag{43}$$

for constraint matrices $\mathbf{A}^i \in \mathbf{S}^n$. The reformulation requires converting the constraints and introducing appropriate slack variables. The constraints for our SDP are particularly sparse which makes it inefficient to use standard packages like CVXOPT and SDPA [21, 24]. Instead, we use the alternating direction method of [50] which is specifically designed for SDP problems with sparse structure and orthogonal constraints. The pseudocode appears in Algorithm 1.

**Algorithm 1** Alternating direction augmented Lagrangian method [50].

**Require:** Constraint matrices $\mathbf{A}, \mathbf{C} \in \mathbf{S}^n$, constraint vector $\mathbf{b} \in \mathbb{R}^m$, iterations $T$, tolerance $t \geq 0$

**Ensure:** Output $\mathbf{X}$ approximately satisfies Equation (42)

$\mathbf{X}^0 \leftarrow \mathbf{0}_{n\times n}$ ▷ Zero matrix

$\mathbf{S}^0 \leftarrow \mathbf{I}_{n\times n}$

**for** $k = 0, 1, \ldots, T-1$ **do**

$y^{k+1} \leftarrow -(\mathcal{A}\mathcal{A}^*)^+(\mathcal{A}(\mathbf{X}^k) - \mathbf{b} + \mathcal{A}(\mathbf{S}^k - \mathbf{C}))$ ▷ $(\cdot)^+$ denotes Moore-Penrose pseudoinverse

$\mathbf{V}^{k+1} \leftarrow \mathbf{C} - \mathcal{A}^*(y^{k+1}) - \mathbf{X}^k$

$\mathbf{S}^{k+1} \leftarrow \mathbf{Q}_+\mathbf{\Sigma}_+\mathbf{Q}_+^\top$ ▷ $\mathbf{Q}_+, \mathbf{\Sigma}_+$ contain the non-negative eigendecomposition of $\mathbf{V}^{k+1}$

$\mathbf{X}^{k+1} \leftarrow \mathbf{S}^{k+1} - \mathbf{V}^{k+1}$

$\mathbf{X}^{k+1} \leftarrow \operatorname{round}(\mathbf{X}^{k+1})$ ▷ For entries within $t$ of 0 or 1, round to nearest integer

**end for**

$\mathbf{X} \leftarrow \mathbf{X}^{T-1}$

We slightly adapt Algorithm 1 to our problem: We store each matrix in a sparse format, round entries of the solution after every iteration, and use the Moore-Penrose pseudoinverse.

The first natural question is how well Algorithm 1 performs. We test this by generating random Boolean functions $f : X \to \{0, 1\}$ where $X \subseteq \{0, 1\}^n$ for different values of $n$. Since the dimension of the SDP grows exponentially with $|X|$, we fix $|X| = 32$ for our experiments. For each instance, we compute the maximum numerical error

$$\varepsilon \coloneqq \max_{x,y:f(x)\neq f(y)} |\langle\psi_x|\phi_y\rangle|. \tag{44}$$

The results in Theorem 16 depend on the error $\varepsilon$ being small so it's important that we can obtain solutions with small error efficiently. Figure 1 shows how $\varepsilon$ decreases with the number of iterations $T$ of the SDP solver.

The next question is whether the error is sufficiently small to satisfy the requirements of Theorem 16. Figure 2 shows how $\varepsilon$ is typically much smaller than the bound required in Theorem 16 for random functions of up to 25 bits with domain size 32. Our experiments test for whether $\varepsilon$ satisfies the bound of Theorem 16 in the case that $\kappa^* = \kappa$. It could be that even larger $\varepsilon$ is tolerated by considering $\kappa^* < \kappa$.

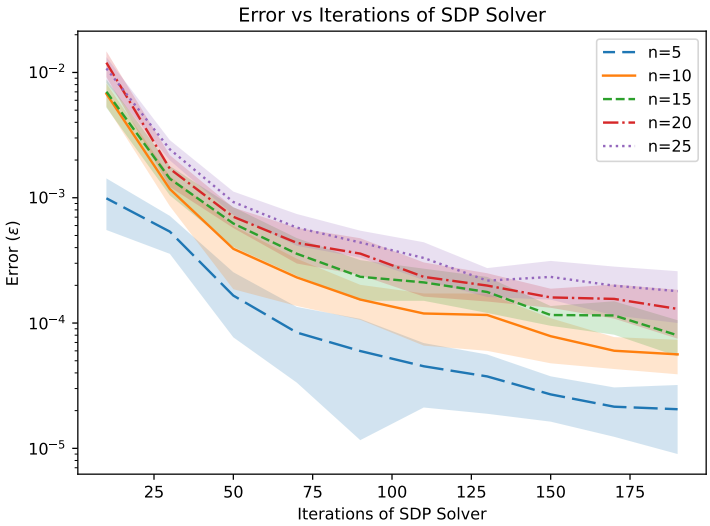



**Figure 1** For random Boolean functions with a domain of fixed size, the maximum numerical error decreases with the number of iterations of the SDP solver. Note: the vertical axis is on a logarithmic scale and the shaded regions contain one standard deviation from 20 random instances.

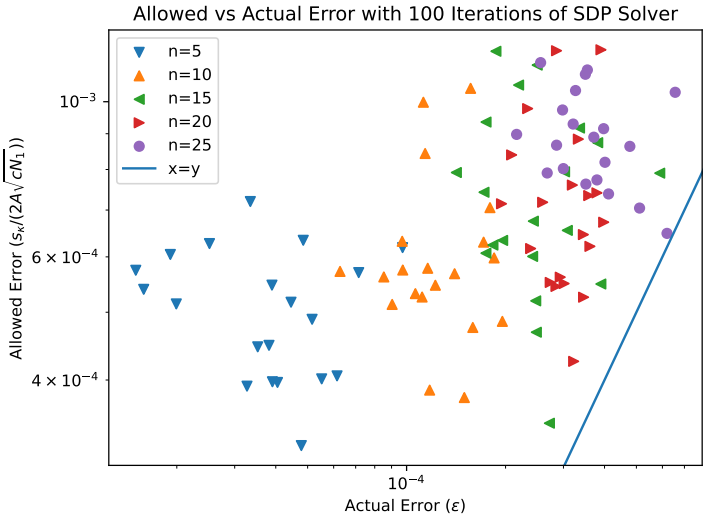



**Figure 2** For random Boolean functions with a domain of fixed size, the maximum numerical error stays below the threshold required to construct a provably correct bounded-error quantum algorithm. Note: both axes are on a logarithmic scale and the green region above the diagonal line indicates the error is small enough.

## References


1 Scott Aaronson. Open problems related to quantum query complexity. *ACM Transactions on Quantum Computing*, 2(4):1–9, 2021.

2 Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 636–643, 2000.

3 Noel T Anderson, Jay-U Chung, Shelby Kimmel, Da-Yeon Koh, and Xiaohan Ye. Improved quantum query complexity on easier inputs. *arXiv preprint*, 2023. arXiv:2303.00217.

4 Jeongho Bang, Junghee Ryu, Seokwon Yoo, Marcin Pawłowski, and Jinhyoung Lee. A strategy for quantum algorithm design assisted by machine learning. *New Journal of Physics*, 16(7):073017, 2014.

5 Howard Barnum, Michael Saks, and Mario Szegedy. Quantum query complexity and semi-definite programming. In *18th IEEE Annual Conference on Computational Complexity, 2003. Proceedings.*, pages 179–193. IEEE, 2003.

6 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001.

7 Salman Beigi and Leila Taghavi. Quantum Speedup Based on Classical Decision Trees. *arXiv*, September 2019. arXiv:1905.13095.

8 Salman Beigi and Leila Taghavi. Span Program for Non-binary Functions. *arXiv*, May 2019. arXiv:1805.02714.

9 Aleksandrs Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 207–216. IEEE, 2012.

10 Aleksandrs Belovs and Duyal Yolcu. One-way ticket to las vegas and the quantum adversary. *arXiv preprint*, 2023. arXiv:2301.02003.

11 Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness. *Quantum Information & Computation*, 18(1-2):18–50, February 2018.

12 Titouan Carette, Mathieu Laurière, and Frédéric Magniez. Extended learning graphs for triangle finding. *Algorithmica*, 82(4):980–1005, 2020.

13 Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

14 Andrew Childs. Lecture notes on quantum algorithms, 2022. URL: https://www.cs.umd.edu/~amchilds/qa/qa.pdf.

15 Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 59–68, 2003.

16 R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, January 1998. doi:10.1098/rspa.1998.0164.

17 Michael Czekanski, Shelby Kimmel, and R. Teal Witter. Robust and space-efficient dual adversary quantum query algorithms, 2023. arXiv:2306.15040.

18 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

19 J Niel De Beaudrap, Richard Cleve, John Watrous, et al. Sharp quantum versus classical query complexity separations. *Algorithmica*, 34(4):449–461, 2002.

20 Kai DeLorenzo, Shelby Kimmel, and R. Teal Witter. Applications of the Quantum Algorithm for st-Connectivity. In Wim van Dam and Laura Mancinska, editors, *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TQC.2019.6.

21 Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

22 Lijun Ding, Alp Yurtsever, Volkan Cevher, Joel A Tropp, and Madeleine Udell. An optimal-storage approach to semidefinite programming using approximate complementarity. *SIAM Journal on Optimization*, 31(4):2695–2725, 2021.

23 Simon Foucart, Holger Rauhut, Simon Foucart, and Holger Rauhut. *An invitation to compressive sensing*. Springer, 2013.

24 Katsuki Fujisawa, Masakazu Kojima, Kazuhide Nakata, and Makoto Yamashita. Sdpa (semidefinite programming algorithm) user's manual—version 6.2. 0. *Department of Mathematical and Com-puting Sciences, Tokyo Institute of Technology. Research Reports on Mathematical and Computing Sciences Series B: Operations Research*, 2002.

25 Dmitry Gavinsky, Julia Kempe, and Ronald De Wolf. Strengths and weaknesses of quantum fingerprinting. In *21st Annual IEEE Conference on Computational Complexity (CCC'06)*, pages 8–pp. IEEE, 2006.

26 Lov K. Grover. Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Physical Review Letters*, 79(2):325–328, July 1997. doi:10.1103/PhysRevLett.79.325.

27 Aram W Harrow, Ashley Montanaro, and Anthony J Short. Limitations on quantum dimensionality reduction. *International Journal of Quantum Information*, 13(04):1440001, 2015.

28 Peter Hoyer, Troy Lee, and Robert Spalek. Negative weights make adversaries stronger. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 526–535, 2007.

29 IBM. The IBM quantum development roadmap, 2022. URL: https://www.ibm.com/quantum/roadmap.

30 Tsuyoshi Ito and Stacey Jeffery. Approximate Span Programs. *Algorithmica*, 81(6):2158–2195, June 2019. doi:10.1007/s00453-018-0527-1.

31 Stacey Jeffery. Span programs and quantum space complexity. *arXiv preprint*, 2019. arXiv:1908.04232.

32 Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. Quantum algorithm for path-edge sampling. *arXiv preprint*, 2023. arXiv:2303.03319.

33 Stacey Jeffery and Sebastian Zur. Multidimensional quantum walks, with application to $k$-distinctness. *arXiv preprint*, 2022. arXiv:2208.13492.

34 William B Johnson and Joram Lindenstraus. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.

35 Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *Journal of the ACM (JACM)*, 61(1):1–23, 2014.

36 Mauricio Karchmer and Avi Wigderson. On span programs. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 102–111. IEEE, 1993.

37 A. Yu Kitaev. Quantum measurements and the Abelian Stabilizer Problem. *arXiv*, November 1995. arXiv:quant-ph/9511026.

38 K. G. Larsen and J. Nelson. Optimality of the Johnson-Lindenstrauss Lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638, October 2017. doi:10.1109/FOCS.2017.64.

39 Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638. IEEE, 2017.

40 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum Query Complexity of State Conversion. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 344–353, October 2011. doi:10.1109/FOCS.2011.75.

41 Daniel Nagaj, Pawel Wocjan, and Yong Zhang. Fast amplification of qma. *arXiv preprint*, 2009. arXiv:0904.1549.

42 Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5(1):4213, 2014.

43 Ben W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 544–551, October 2009. doi:10.1109/FOCS.2009.55.

44 Ben W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the 2011 Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings, pages 560–569. Society for Industrial and Applied Mathematics, January 2011. doi:10.1137/1.9781611973082.44.

45 Ben W Reichardt and Robert Spalek. Span-program-based quantum algorithm for evaluating formulas. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 103–112, 2008.

46 B.W. Reichardt. Span programs are equivalent to quantum query algorithms. *SIAM Journal on Computing*, 43(3):1206–1219, 2014. doi:10.1137/100792640.

47 Anthony Man-Cho So, Yinyu Ye, and Jiawei Zhang. A unified theorem on sdp rank reduction. *Mathematics of Operations Research*, 33(4):910–920, 2008.

48 Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.

49 Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

50 Zaiwen Wen, Donald Goldfarb, and Wotao Yin. Alternating direction augmented lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2(3-4):203–230, 2010.

# Revisiting the Random Subset Sum Problem

**Arthur Carvalho Walraven Da Cunha**
Université Côte d'Azur, Inria Sophia Antipolis, CNRS, France

**Francesco d'Amore**
Aalto University, Finland
Université Côte d'Azur, Inria Sophia Antipolis, CNRS, France

**Frédéric Giroire**
Université Côte d'Azur, Inria Sophia Antipolis, CNRS, France

**Hicham Lesfari**
Université Côte d'Azur, Inria Sophia Antipolis, CNRS, France

**Emanuele Natale**
Université Côte d'Azur, Inria Sophia Antipolis, CNRS, France

**Laurent Viennot**
Inria Paris, IRIF, France

## Abstract

The average properties of the well-known *Subset Sum Problem* can be studied by means of its randomised version, where we are given a target value $z$, random variables $X_1, \ldots, X_n$, and an error parameter $\varepsilon > 0$, and we seek a subset of the $X_i$s whose sum approximates $z$ up to error $\varepsilon$. In this setup, it has been shown that, under mild assumptions on the distribution of the random variables, a sample of size $\mathcal{O}(\log(1/\varepsilon))$ suffices to obtain, with high probability, approximations for all values in $[-1/2, 1/2]$. Recently, this result has been rediscovered outside the algorithms community, enabling meaningful progress in other fields. In this work, we present an alternative proof for this theorem, with a more direct approach and resourcing to more elementary tools.

**2012 ACM Subject Classification** Mathematics of computing → Stochastic processes; Mathematics of computing → Combinatoric problems

**Keywords and phrases** Random subset sum, Randomised method, Subset-sum, Combinatorics

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.37

**Related Version** *arXiv Preprint*: `https://arxiv.org/abs/2204.13929`

**Funding** *Frédéric Giroire*: Supported by the French government through the UCA JEDI (ANR-15-IDEX-01) and EUR DS4H (ANR-17-EURE-004) Investments in the Future projects, by the PEPR CARECLOUD, and by the European Project dAIEDGE.
*Emanuele Natale*: Supported by the AID INRIA-DGA agreement n°2019650072.

**Acknowledgements** The authors are thankful to Bianca C. Araújo and the paper reviewers for the feedback on the presentation of the paper. The authors are also grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support.

## 1 Introduction

In the *Subset Sum Problem (SSP)*, one is given as input a set of $n$ integers $X = \{x_1, x_2, \ldots, x_n\}$ and a target value $z$, and wishes to decide if there exists a subset of $X$ that sums to $z$. That is, one is to reason about a subset $S \subseteq [n]$ such that $\sum_{i \in S} x_i = z$. The special case where $z$ is half of the sum of $X$ is known as the *Number Partition Problem (NPP)*. The converse reduction is also rather immediate.[1]

[1] To find a subset of $X$ summing to $z$, one only needs to solve the NPP for the set $X \cup \{2z, \sum_{i \in [n]} x_i\}$. By doing so, one of the parts must consist of the element $\sum_{i \in [n]} x_i$ alongside the desired subset.





Be it in either of these forms, the SSP finds applications in a variety of fields, ranging from combinatorial number theory [31] to cryptography [20, 26]. In complexity theory, the SSP is a well-known NP-complete problem, being a common base for NP-completeness proofs. In fact, the NPP version figures among Garey and Johnson's six basic NP-hard problems [19]. Under certain circumstances, the SSP can be challenging even for heuristics that perform well for many other NP-hard problems [25, 30], and a variety of dedicated algorithms have been proposed to solve it [10, 17, 22–24]. Nonetheless, it is not hard to solve it in polynomial time if we restrict the input integers to a fixed range [5]. It suffices to recursively list all achievable sums using the first $i$ integers: we start with $A_0 = \{0\}$ and compute $A_{i+1}$ as $A_i \cup \{a + x_{i+1} \mid a \in A_i\}$. For integers in the range $[0, R]$, the search space has size $\mathcal{O}(nR)$.

Studying how the problem becomes hard as we consider larger ranges of integers (relative to $n$) requires a randomised version of the problem, the *Random Subset Sum Problem (RSSP)*, where the input values are taken as independently and identically distributed random variables. In this setup, the work [6] proved that the problem experiences a phase transition in its average complexity as the range of integers increases.

The result we approach in this work comes from related studies on the typical properties of the problem. In [27] the author proves that, under fairly general conditions, the expected minimal distance between a subset-sum and the target value is exponentially small. More specifically, they show the following result.

▶ **Theorem 1** (Lueker, 1998). *Let $X_1, \ldots, X_n$ be independent uniform random variables over $[-1, 1]$, and let $\varepsilon \in (0, 1/3)$. There exists a universal constant $C > 0$ such that, if $n \geq C \log(1/\varepsilon)$, then, with probability at least $1 - \varepsilon$, for all $z \in [-1, 1]$ there exists $S_z \subseteq [n]$ for which*

$$\left| z - \sum_{i \in S_z} X_i \right| \leq \varepsilon.$$

That is, a rather small number (of the order of $\log \frac{1}{\varepsilon}$) of random variables suffices to have a high probability of approximating not only a single target $z$, but all values in an interval.

Even though Theorem 1 is stated and proved for uniform random variables and target values in $[-1, 1]$, it is not hard to extend the result to a broad class of distributions[2] and a wider range of targets. This generality makes the theorem a powerful tool for the analysis of random structures and has recently proven to be particularly useful in the field of Machine Learning, taking part in a proof of the Strong Lottery Ticket Hypothesis [29] and in subsequent related works [11, 13, 14, 18], and in Federated Learning [32].

Generalisations of the RSSP have played important roles in the study of random Knapsack problems [3,4], and to random binary integer programs [7,8]. In particular, the works [2,7,8,14] recently provided an extension of Theorem 1 to multiple dimensions. As for the equivalent Random Number Partitioning Problem, [12] recently generalised [6] and the integer version of the RSSP to non-binary integer coefficients.

The simplicity and ubiquity of the SSP have granted the related results a special didactic place. Be it as a first example of an NP-complete problem [19], a path to science communication [21], or simply as a frame for the demonstration of advanced techniques [28], it has been a tool to make important, but sometimes complicated, ideas easier to communicate.

[2] Distributions whose probability density function $f$ satisfies $f(x) \geq b$ for all $x \in [-a, a]$, for some constants $a, b > 0$ (see Corollary 3.3 from [27]).

This work offers a substantially simpler alternative to the original proof of Theorem 1 by following a general framework introduced in the context of the analysis of Rumour Spreading algorithms [15]. Originally, the work [27] approaches Theorem 1 by considering the random variable associated with the proportion of the values in the interval $[-1, 1]$ that can be approximated up to error $\varepsilon$ by the sum of some subset of the first $t$ variables, $X_1, \ldots, X_t$.

After restricting to some specific types of subsets, they proceed to evaluate the expected per-round growth of this proportion, conditioned on the outcomes of $X_1, \ldots, X_t$. Their strategy is to analyse this expected increase by martingale theory, which only becomes possible after a non-linear transformation of the variables of interest. Those operations hinder any intuition for the obtained martingale. Nonetheless, a subsequent application of the Azuma-Hoeffding bound [1] followed by a case analysis leads to the result.

The argument presented here starts in the same direction as the original one, tracking the mass of values with suitable approximations as we reveal the values of the random variables $X_1, \ldots, X_n$ one by one. However, we quickly diverge from [27], managing to obtain an estimation of the expected growth of this mass without discarding any subset-sum. We eventually restrict the argument to some types of subsets, but we do so at a point where the need for such restriction is clear.

We proceed to directly analyse the estimation obtained, without any transformations. Following [15], this estimation reveals two expected behaviours in expectation, which can be analysed similarly: as we consider the first variables, the proportion of approximated values grows very fast; then, after a certain point, the proportion of non-approximable values decreases very fast.

We remark that, while Theorem 1 crucially relies on tools from martingale theory such as Azuma-Hoeffding's inequality, which are not part of standard Computer Science curricula, our argument makes use of much more elementary results[3] which should make it accessible enough for an undergraduate course on randomised algorithms.

## 2 Our argument

In this section, we provide an alternative argument for proving Theorem 1. It takes shape much like the pseudo-polynomial algorithm we described in the introduction. Leveraging the recursive nature of the problem, we construct a process which, at time $t$, describes the proportion of the interval $[-1, 1]$ that can be approximated by some subset of the first $t$ variables.

We will show that with a suitable number of uniform variables (proportional to $\log(1/\varepsilon)$) a factor of $1 - \varepsilon/2$ of the values in $[-1, 1]$ can be approximated up to error $\varepsilon$. This implies that any $z \in [-1, 1]$ which cannot be approximated within error $\varepsilon$ is at most $\varepsilon$ away from a value that can. Therefore it is possible to approximate $z$ up to error $2\varepsilon$.

### 2.1 Preliminaries

Let $X_1, \ldots, X_n$ be realisations of random variables as in Theorem 1, and, without loss of generality, fix $\varepsilon > 0$. We say a value $z \in \mathbb{R}$ is *$\varepsilon$-approximated at time $t$* if and only if there exists $S \subseteq [t]$ such that $|z - \sum_{i \in S} X_i| < \varepsilon$. For $0 \leq t \leq n$, let $f_t \colon \mathbb{R} \to \{0, 1\}$ be the indicator function for the event "$z$ is $\varepsilon$-approximated at time $t$". Therefore, we have $f_0 = \mathbb{1}_{(-\varepsilon, \varepsilon)}$, since only the interval $(-\varepsilon, \varepsilon)$ can be approximated by an empty set of values. From there, we can

[3] Namely, the intermediate value theorem, Markov's inequality, and standard Hoeffding bounds.

exploit the recurrent nature of the problem: a value $z$ can be $\varepsilon$-approximated at time $t+1$ if and only if either $z$ or $z - X_{t+1}$ could already be approximated at time $t$. This implies that for all $z \in \mathbb{R}$ we have that

$$f_{t+1}(z) = f_t(z) + \big(1 - f_t(z)\big)\, f_t(z - X_{t+1}). \tag{1}$$

To keep track of the proportion of values in $[-1, 1]$ that can be $\varepsilon$-approximated at each step, we define, for each $0 \le t \le n$, the random variable

$$v_t = \frac{1}{2}\int_{-1}^{1} f_t(z)\,\mathrm{d}z.$$

For better readability, throughout the text we will refer to $v_t$ simply as "the volume."

As we mentioned, it suffices to show that, with high probability, at time $n$, enough of the interval is $\varepsilon$-approximated (more precisely, that $v_n \ge 1 - \varepsilon/2$) to conclude that the entire interval is $2\varepsilon$-approximated.

### 2.1.1 Expected behaviour

Our first lemma provides a lower bound on the expected value of $v_t$.

▶ **Lemma 2.** *For all $0 \le t < n$, it holds that*

$$\mathbb{E}\big[v_{t+1} \,\big|\, X_1, \ldots, X_t\big] \ge v_t \left[1 + \frac{1}{4}\,(1 - v_t)\right].$$

**Proof.** The definition of $v_t$ and the recurrence in Equation (1) give us that

$$\begin{aligned}
\mathbb{E}\big[v_{t+1} \,\big|\, X_1, \ldots, X_t\big] &= \mathbb{E}\left[\frac{1}{2}\int_{-1}^{1} f_{t+1}(z)\,\mathrm{d}z \,\middle|\, X_1, \ldots, X_t\right] \\
&= \int_{-1}^{1} \frac{1}{2}\left(\frac{1}{2}\int_{-1}^{1} f_t(z) + \big(1 - f_t(z)\big)\, f_t(z - x)\,\mathrm{d}z\right)\mathrm{d}x \\
&= \frac{1}{2}\int_{-1}^{1} f_t(z)\,\mathrm{d}z \int_{-1}^{1} \frac{1}{2}\,\mathrm{d}x + \frac{1}{2}\int_{-1}^{1}\frac{1}{2}\int_{-1}^{1} \big(1 - f_t(z)\big)\, f_t(z - x)\,\mathrm{d}z\,\mathrm{d}x \\
&= v_t + \frac{1}{4}\int_{-1}^{1} \big(1 - f_t(z)\big) \int_{-1}^{1} f_t(z - x)\,\mathrm{d}x\,\mathrm{d}z \\
&= v_t + \frac{1}{4}\int_{-1}^{1} \big(1 - f_t(z)\big) \int_{z-1}^{z+1} f_t(y)\,\mathrm{d}y\,\mathrm{d}z,
\end{aligned}$$

where the last equality holds by substituting $y = z - x$. For the previous ones, we apply the basic properties of integrals and Fubini's theorem to change the order of integration.

We now look for a lower bound for the last integral in terms of $v_t$. To this end, we exploit that, since all integrands are non-negative, for all $u \in [-1/2, 1/2]$ we have that

$$\begin{aligned}
\int_{-1}^{1} \big(1 - f_t(z)\big) \int_{z-1}^{z+1} f_t(y)\,\mathrm{d}y\,\mathrm{d}z &\ge \int_{u-\frac{1}{2}}^{u+\frac{1}{2}} \big(1 - f_t(z)\big) \int_{z-1}^{z+1} f_t(y)\,\mathrm{d}y\,\mathrm{d}z \\
&\ge \int_{u-\frac{1}{2}}^{u+\frac{1}{2}} \big(1 - f_t(z)\big) \int_{u-\frac{1}{2}}^{u+\frac{1}{2}} f_t(y)\,\mathrm{d}y\,\mathrm{d}z.
\end{aligned}$$

Both inequalities come from range restrictions: in the first, we use that $u \in [-1/2, 1/2]$ implies $[u-1/2, u+1/2] \subseteq [-1, 1]$; for the second, we have that $[u-1/2, u+1/2] \subseteq [z-1, z+1]$ for all $z \in [u-1/2, u+1/2]$.

To relate the expression to $v_t$ explicitly, we choose $u$ in a way that the window $[u-1/2, u+1/2]$ entails exactly half of $v_t$. The existence of such $u$ may become clear by recalling the definition of $v_t$. To make it formal, consider the function given by

$$h(u) = \frac{1}{2} \int_{u-\frac{1}{2}}^{u+\frac{1}{2}} f_t(y)\,\mathrm{d}y,$$

and observe that

$$\min\{h(-1/2),\, h(1/2)\} \leq \frac{v_t}{2}, \qquad \text{and} \qquad \max\{h(-1/2),\, h(1/2)\} \geq \frac{v_t}{2}.$$

Thus, by the intermediate value theorem, there exists $u^* \in [-1/2, 1/2]$ for which $h(u^*) = v_t/2$, that is, for which

$$\frac{1}{2} \int_{u^*-\frac{1}{2}}^{u^*+\frac{1}{2}} f_t(y)\,\mathrm{d}y = \frac{v_t}{2}.$$

Altogether, we can conclude that

$$\begin{aligned}
\mathbb{E}\left[v_{t+1} \,\middle|\, X_1, \ldots, X_t\right] &= v_t + \frac{1}{4} \int_{-1}^{1} \left(1 - f_t(z)\right) \int_{z-1}^{z+1} f_t(y)\,\mathrm{d}y\,\mathrm{d}z \\
&\geq v_t + \frac{1}{2} \int_{u^*-\frac{1}{2}}^{u^*+\frac{1}{2}} \left(1 - f_t(z)\right) \left(\frac{1}{2} \int_{u^*-\frac{1}{2}}^{u^*+\frac{1}{2}} f_t(y)\,\mathrm{d}y\right) \mathrm{d}z \\
&= v_t + \left(\frac{1}{2} - \frac{v_t}{2}\right) \frac{v_t}{2} \\
&= v_t \left[1 + \frac{1}{4}\left(1 - v_t\right)\right].
\end{aligned}$$

◀

Lemma 2 tells us that, if $v_t$ were to behave as expected, it should grow exponentially up to $1/2$, at which point $1 - v_t$ starts to decrease exponentially. The rest of the proof follows accordingly, with Section 2.2 analysing the progress of $v_t$ up to one half, and Section 2.3 analogously following the complementary value, $1 - v_t$, starting from one half. By building on the results from Section 2.2, we obtain fairly straightforward proofs in Section 2.3. Thus, the following subsection comprises the core of our argument.

## 2.2 Growth of the volume up to 1/2

Arguably, the main challenge in analysing the RSSP is the existence of over-time dependencies and deciding how to overcome it sets much of the course the proof will take. Our strategy consists in constructing another process which dominates the original one while being free of dependencies.

Let $\tau_1$ be the first time at which the volume exceeds $1/2$, that is, let

$$\tau_1 = \min\{t \geq 0 : v_t > 1/2\}.$$

We just proved that up to time $\tau_1$ the process $v_t$ enjoys exponential growth in expectation. In the following lemma, we apply a basic concentration inequality to translate this property into a constant probability of exponential growth for $v_t$ itself.

▶ **Lemma 3.** *Given $\beta \in (0, 1/8)$, let $p_\beta = 1 - \frac{7}{8(1-\beta)}$. For all integers $0 \leq t < \tau_1$ it holds that*

$$\Pr\left[v_{t+1} \geq v_t(1+\beta) \,\middle|\, X_1, \ldots, X_t,\, t < \tau_1\right] \geq p_\beta.$$

**Proof.** The result shall follow easily from reverse Markov's inequality [9, Lemma 4] and the bound from Lemma 2. However, doing so requires a suitable upper bound on $v_{t+1}$ and, while $2v_t$ would serve the purpose, such bound does not hold in general.

We overcome this limitation by fixing $t$ and considering how much $v_t$ would grow in the next step if we were to consider only values $\varepsilon$-approximated at time $t$ that happen to lie in $[-1, 1]$ after being translated by $X_{t+1}$. Making it precise by the means of the recurrence in Equation (1), we define

$$\tilde{v} = \frac{1}{2}\int_{-1}^{1} \Big[f_t(z) + \big(1 - f_t(z)\big)\, f_t(z - X_{t+1}) \cdot \mathbb{1}_{[-1,1]}(z - X_{t+1})\Big]\, \mathrm{d}z.$$

This expression differs from the one for $v_{t+1}$ only by the inclusion of the characteristic function of $[-1, 1]$. This not only implies that $\tilde{v} \leq v_{t+1}$, but also that $\tilde{v}$ can replace $v_{t+1}$ in the bound from Lemma 2, since the argument provided there eventually restricts itself to integrals within $[-1, 1]$, trivialising $\mathbb{1}_{[-1,1]}$. Moreover, as we obtain $\tilde{v}$ without the influence of values from outside $[-1, 1]$, we must have $\tilde{v} \leq 2v_t$. Finally, using that $t < \tau_1$ implies $v_t < 1/2$ and chaining the previous conclusions in respective order , we conclude that

$$\begin{aligned}
\Pr\left[v_{t+1} \geq v_t(1+\beta) \,\middle|\, X_1, \ldots, X_t, t < \tau_1\right] &\geq \Pr\left[\tilde{v} \geq v_t(1+\beta) \,\middle|\, X_1, \ldots, X_t, t < \tau_1\right] \\
&\geq \frac{\mathbb{E}[\tilde{v} \,|\, X_1, \ldots, X_t, t < \tau_1] - v_t(1+\beta)}{2v_t - v_t(1+\beta)} \\
&\geq \frac{\frac{9}{8}v_t - v_t(1+\beta)}{2v_t - v_t(1+\beta)} \\
&= 1 - \frac{7}{8(1-\beta)},
\end{aligned}$$

where we applied the reverse Markov's inequality in the second step. ◀

The previous lemma naturally leads us to look for bounds on $\tau_1$, that is, to estimate the time needed for the process to reach volume $1/2$. As expected, the exponential nature of the process yields a logarithmic bound.

▶ **Lemma 4.** *Let $t$ be an integer and given $\beta \in (0, 1/8)$, let $p_\beta = 1 - \frac{7}{8(1-\beta)}$ and $i^* = \left\lceil \frac{\log \frac{1}{2\varepsilon}}{\log(1+\beta)} \right\rceil$. If $t \geq i^*/p_\beta$, then*

$$\Pr\left[\tau_1 \leq t\right] \geq 1 - \exp\left[-\frac{2p_\beta^2}{t}\left(t - \frac{i^*}{p_\beta}\right)^2\right].$$

**Proof.** The main idea behind the proof is to define a new random variable which stochastically dominates $\tau_1$ while being simpler to analyse. We begin by discretising the domain $(0, 1/2]$ of the volume into sub-intervals $\{I_i\}_{0 \leq i \leq i^*}$ defined as follows:

$$\begin{cases}
I_0 = (0, \varepsilon], \\
I_i = \left(\varepsilon(1+\beta)^{i-1},\, \varepsilon(1+\beta)^i\right] \text{ for } 1 \leq i < i^*, \\
I_{i^*} = \left(\varepsilon(1+\beta)^{i^*-1},\, \frac{1}{2}\right],
\end{cases}$$

where $i^*$ is the smallest integer for which $\varepsilon\,(1+\beta)^{i^*} \geq 1/2$, that is, $i^* = \left\lceil \frac{\log \frac{1}{2\varepsilon}}{\log(1+\beta)} \right\rceil$.

Now, for each $i \geq 0$, we direct our interest to the number of steps required for $v_t$ to exit the sub-interval $I_i$ after first entering it. By Lemma 3, this amount is majorised by a geometric random variable $Y_i \sim \text{Geom}(p_\beta)$. Therefore, we can conclude that $\tau_1$ is stochastically dominated by the sum of such variables, that is, for $t \in \mathbb{N}$, we have that

$$\Pr[\tau_1 \geq t] \leq \Pr\left[\sum_{i=1}^{i^*} Y_i \geq t\right]. \tag{2}$$

Let $B_t \sim \text{Bin}(t, p_\beta)$ be a binomial random variable. For the sum of geometric random variables, it holds that $\Pr\left[\sum_{i=1}^{i^*} Y_i \leq t\right] = \Pr[B_t \geq i^*]$. Since $\mathbb{E}[B_t] = tp_\beta$, the Hoeffding bound for binomial random variables [16, Theorem 1.1] implies that, for all $\lambda \geq 0$, we have that $\Pr[B_t \leq tp_\beta - \lambda] \leq \exp(-2\lambda^2/t)$. Setting $t$ such that $tp_\beta - \lambda = i^*$, we obtain that

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{i^*} Y_i \geq t\right] &\leq \Pr[B_t \leq i^*] \\
&\leq \exp\left[-\frac{2}{t}\left(tp_\beta - i^*\right)^2\right] \\
&= \exp\left[-\frac{2p_\beta^2}{t}\left(t - \frac{i^*}{p_\beta}\right)^2\right],
\end{aligned}$$

which holds as long as $\lambda = tp_\beta - i^* \geq 0$, that is, for all $t \geq \frac{1}{p_\beta}\left\lceil \frac{\log \frac{1}{2\varepsilon}}{\log(1+\beta)} \right\rceil$.

The thesis follows by applying this to Equation (2) and considering the complementary events. ◀

Since we are done with the analysis of the first phase, we can fix the value of $\beta$ and rearrange the bound in Lemma 4 to make it easier to apply later.

▶ **Corollary 5.** *Let $\varepsilon \in (0, \frac{1}{3})$, and let $t$ be an integer satisfying $t \geq 264 \log \frac{1}{\varepsilon}$. Then*

$$\Pr[\tau_1 \leq t] \geq 1 - \exp\left[-\frac{2}{225t}\left(t - 264 \log \frac{1}{\varepsilon}\right)^2\right].$$

**Proof.** Setting $\beta = \frac{1}{16}$ in Lemma 4 and, thus, $p_\beta = \frac{1}{15}$, it suffices to notice that

$$\frac{15 \log \frac{1}{2\varepsilon}}{\log \frac{17}{16}} + 15 \leq 264 \log \frac{1}{\varepsilon}.$$ ◀

## 2.3 Growth of the volume from 1/2

Here we study the second half of the process: from the moment the volume reaches $1/2$ up to the time it gets to $1 - \varepsilon/2$. We do so by analysing the complementary stochastic process, i.e., by tracking, from time $\tau_1$ onwards, the proportion of the interval $[-1, 1]$ that cannot be approximated up to error $\varepsilon$. More precisely, we consider the process $\{w_t\}_{t \geq 0}$, defined by $w_t = 1 - v_{\tau_1 + t}$.

We shall obtain results for $w_t$ similar to those we have proved for $v_t$. Fortunately, building on the previous results makes those proofs quite straightforward. We start by noting that a statement analogous to Lemma 2 follows immediately from the definition of $w_{t+1}$ and Lemma 2.

▶ **Corollary 6.** *For all $t \geq 0$, it holds that*

$$\mathbb{E}\big[w_{t+1} \,\big|\, X_1, \ldots, X_{\tau_1+t}\big] \leq w_t \left[1 - \frac{1}{4}\,(1 - w_t)\right].$$

Let $\tau_2$ the first time that $w_t$ gets smaller than or equal to $\varepsilon/2$, that is, let

$$\tau_2 = \min\left\{t \geq 0 : w_t \leq \varepsilon/2\right\}.$$

The following lemma bounds this quantity, in analogy to Lemma 4.

▶ **Lemma 7.** *For all $t > 0$, it holds that*

$$\Pr\left[\tau_2 \leq t\right] \geq 1 - \exp\left[-\frac{1}{8}\left(t - 8\log\frac{1}{\varepsilon}\right)\right].$$

**Proof.** Applying that $1 - w_t = v_{\tau_1+t} > 1/2$ to Corollary 6 gives the bound

$$\mathbb{E}\big[w_{t+1} \,\big|\, X_1, \ldots, X_{\tau_1+t}\big] \leq \frac{7}{8} w_t. \tag{3}$$

Moreover, from the conditional expectation theory, for any two random variables $X$ and $Y$, we have $\mathbb{E}\big[\mathbb{E}[X \,|\, Y]\big] = \mathbb{E}[X]$. From this and Equation (3), we can conclude that

$$\mathbb{E}[w_t] = \mathbb{E}\Big[\mathbb{E}\big[w_t \,\big|\, X_1, \ldots, X_{\tau_1+t-1}\big]\Big] \leq \frac{7}{8}\,\mathbb{E}\big[w_{t-1}\big],$$

which, by recursion, yields that

$$\mathbb{E}[w_t] \leq \left(\frac{7}{8}\right)^t \mathbb{E}[w_0] \leq \frac{1}{2}\left(\frac{7}{8}\right)^t.$$

Finally, by Markov's inequality,

$$\begin{aligned}\Pr\left[\tau_2 \geq t\right] &\leq \Pr\left[w_t \geq \frac{\varepsilon}{2}\right] \\ &\leq \frac{2\,\mathbb{E}[w_t]}{\varepsilon} \\ &\leq \frac{1}{\varepsilon}\left(\frac{7}{8}\right)^t,\end{aligned}$$

and, since $\log\frac{8}{7} > \frac{1}{8}$, it holds that

$$\Pr\left[\tau_2 \geq t\right] \leq \exp\left[-\frac{1}{8}\left(t - 8\log\frac{1}{\varepsilon}\right)\right].$$

The thesis follows by considering the complementary event. ◀

## 2.4 Putting everything together

In this section we conclude our argument, finally proving Theorem 1. We first prove a more general statement and then detail how it implies the theorem.

Let $\tau = \tau_1 + \tau_2$, the first time at which the process $\{v_t\}_{t\geq 0}$ reaches at least $1 - \varepsilon/2$.

▶ **Lemma 8.** *Let $\varepsilon \in (0, 1/3)$. There exist constants $C' > 0$ and $\kappa > 0$ such that for every $t \geq C' \log \frac{1}{\varepsilon}$, it holds that*

$$\Pr[\tau \leq t] \geq 1 - 2\exp\left[-\frac{1}{\kappa t}\left(t - C' \log \frac{1}{\varepsilon}\right)^2\right].$$

**Proof.** The definition of $\tau$ allows us to apply Corollary 5 and Lemma 7 quite directly. Indeed, if, for the sake of Corollary 5, we assume that $t/2 \geq 264 \log \frac{1}{\varepsilon}$, we have that

$$\begin{aligned}
\Pr[\tau \leq t] &= \Pr[\tau_1 + \tau_2 \leq t] \\
&\geq \Pr\left[\tau_1 \leq t/2, \tau_2 \leq t/2\right] \\
&\geq \Pr\left[\tau_1 \leq t/2\right] + \Pr\left[\tau_2 \leq t/2\right] - 1 \\
&\geq 1 - \exp\left[-\frac{4}{225t}\left(\frac{t}{2} - 264 \log \frac{1}{\varepsilon}\right)^2\right] - \exp\left[-\frac{1}{8}\left(\frac{t}{2} - 8\log \frac{1}{\varepsilon}\right)\right] \\
&\geq 1 - \exp\left[-\frac{4}{225t}\left(\frac{t}{2} - 264 \log \frac{1}{\varepsilon}\right)^2\right] - \exp\left[-\frac{1}{4t}\left(\frac{t}{2} - 8 \log \frac{1}{\varepsilon}\right)^2\right],
\end{aligned}$$

where the second inequality holds by the union bound. By setting $\kappa = 225$ and $C' = 512$, we obtain the thesis. ◀

The expression in the claim of Lemma 8 can be reformulated as

$$\Pr\left[v_t \geq 1 - \frac{\varepsilon}{2}\right] \geq 1 - 2\exp\left[-\frac{1}{\kappa t}\left(t - C' \log \frac{1}{\varepsilon}\right)^2\right];$$

hence, Theorem 1 follows by taking $C \geq 3C'$ and observing that once we can approximate all but an $\varepsilon/2$ proportion of the interval $[-1, 1]$, any $z \in [-1, 1]$ either is $\varepsilon$-approximated itself, or is at most $\varepsilon$ away from a value that is, which implies that $z$ is $2\varepsilon$-approximated.

## References


1. Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal, Second Series*, 19(3):357–367, 1967.
2. Luca Becchetti, Arthur Carvalho Walraven da Cunha, Andrea Clementi, Francesco d' Amore, Hicham Lesfari, Emanuele Natale, and Luca Trevisan. On the Multidimensional Random Subset Sum Problem. report, Inria & Université Cote d'Azur, CNRS, I3S, Sophia Antipolis, France ; Sapienza Università di Roma, Rome, Italy ; Università Bocconi, Milan, Italy ; Università di Roma Tor Vergata, Rome, Italy, 2022.
3. Rene Beier and Berthold Vöcking. Random knapsack in expected polynomial time. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 232–241. Association for Computing Machinery, 2003. doi:10.1145/780542.780578.
4. Rene Beier and Berthold Vöcking. Probabilistic analysis of knapsack core algorithms. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 468–477. Society for Industrial and Applied Mathematics, 2004.
5. Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
6. Christian Borgs, Jennifer T. Chayes, and Boris G. Pittel. Phase transition and finite-size scaling for the integer partitioning problem. *Random Struct. Algorithms*, 19(3-4):247–288, 2001. doi:10.1002/rsa.10004.
7. Sander Borst, Daniel Dadush, Sophie Huiberts, and Danish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. *CoRR*, abs/2210.05982, 2022. doi:10.48550/arXiv.2210.05982.

8 Sander Borst, Daniel Dadush, Sophie Huiberts, and Samarth Tiwari. On the integrality gap of binary integer programs with Gaussian data. *Mathematical Programming*, 2022. doi:10.1007/s10107-022-01828-1.

9 Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530, 2006.

10 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1777–1796. SIAM, 2021.

11 Rebekka Burkholz, Nilanjana Laha, Rajarshi Mukherjee, and Alkis Gotovos. On the existence of universal lottery tickets. In *International Conference on Learning Representations*, 2022. URL: https://openreview.net/forum?id=SYB4WrJql1n.

12 Xi Chen, Yaonan Jin, Tim Randolph, and Rocco A. Servedio. Average-case subset balancing problems. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 743–778. SIAM, 2022. doi:10.1137/1.9781611977073.33.

13 Arthur da Cunha, Emanuele Natale, and Laurent Viennot. Proving the strong lottery ticket hypothesis for convolutional neural networks. In *International Conference on Learning Representations*, 2022. URL: https://openreview.net/forum?id=Vjki79-619-.

14 Arthur Carvalho Walraven da Cunha, Francesco d'Amore, and Emanuele Natale. Convolutional neural networks contain structured strong lottery tickets. Preprint, June 2023. URL: https://hal.science/hal-04143024.

15 Benjamin Doerr and Anatolii Kostrygin. Randomized rumor spreading revisited. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 138:1–138:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.138.

16 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

17 Andre Esser and Alexander May. Low weight discrete logarithms and subset sum in $2^{0.65n}$ with polynomial memory. *Cryptology ePrint Archive*, 2019.

18 Jonas Fischer and Rebekka Burkholz. Towards strong pruning for lottery tickets with non-zero biases. *CoRR*, abs/2110.11150, 2021. arXiv:2110.11150.

19 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

20 Peter Gemmell and Anna M. Johnston. Analysis of a subset sum randomizer. *IACR Cryptol. ePrint Arch.*, page 18, 2001. URL: http://eprint.iacr.org/2001/018.

21 Brian Hayes. The easiest hard problem. *American Scientist*, 90:113–117, 2002.

22 Alexander Helm and Alexander May. Subset sum quantumly in 1.17^ n. In *13th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

23 Ce Jin, Nikhil Vyas, and Ryan Williams. Fast low-space algorithms for subset sum. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1757–1776. SIAM, 2021.

24 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. *arXiv preprint arXiv:1807.11597*, 2018.

25 David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations research*, 39(3):378–406, 1991.

26 Aniket Kate and Ian Goldberg. Generalizing cryptosystems based on the subset sum problem. *Int. J. Inf. Sec.*, 10(3):189–199, 2011. doi:10.1007/s10207-011-0129-2.

27 George S. Lueker. Exponentially small bounds on the expected optimum of the partition and subset sum problems. *Random Structures and Algorithms*, 12:51–62, 1998.

28 Stephan Mertens. A physicist's approach to number partitioning. *Theor. Comput. Sci.*, 265(1-2):79–108, 2001. doi:10.1016/S0304-3975(01)00153-0.

29 Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris S. Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1b742ae215adf18b75449c6e272fd92d-Abstract.html.

30 Wheeler Ruml, J. Thomas Ngo, Joe Marks, and Stuart M Shieber. Easily searched encodings for number partitioning. *Journal of Optimization Theory and Applications*, 89(2):251–291, 1996.

31 Zhi-Wei Sun. Unification of zero-sum problems, subset sums and covers of z. *Electronic Research Announcements of The American Mathematical Society*, 9:51–60, 2003.

32 Chenghong Wang, Jieren Deng, Xianrui Meng, Yijue Wang, Ji Li, Sheng Lin, Shuo Han, Fei Miao, Sanguthevar Rajasekaran, and Caiwen Ding. A secure and efficient federated learning framework for NLP. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 7676–7682. Association for Computational Linguistics, 2021. doi:10.18653/v1/2021.emnlp-main.606.

# Scheduling with a Limited Testing Budget

## Tight Results for the Offline and Oblivious Settings*

**Christoph Damerius**
Department of Informatics, Universität Hamburg, Germany

**Peter Kling**
Department of Informatics, Universität Hamburg, Germany

**Minming Li**
Department of Computer Science, City University of Hong Kong, China

**Chenyang Xu**
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China

**Ruilong Zhang**[1]
Department of Computer Science and Engineering, University at Buffalo, NY, USA

## Abstract

Scheduling with testing falls under the umbrella of the research on optimization with explorable uncertainty. In this model, each job has an upper limit on its processing time that can be decreased to a lower limit (possibly unknown) by some preliminary action (testing). Recently, Dürr et al. [10] has studied a setting where testing a job takes a unit time, and the goal is to minimize total completion time or makespan on a single machine. In this paper, we extend their problem to the budget setting in which each test consumes a job-specific cost, and we require that the total testing cost cannot exceed a given budget. We consider the offline variant (the lower processing time is known) and the oblivious variant (the lower processing time is unknown) and aim to minimize the total completion time or makespan on a single machine.

For the total completion time objective, we show NP-hardness and derive a PTAS for the offline variant based on a novel LP rounding scheme. We give a $(4+\epsilon)$-competitive algorithm for the oblivious variant based on a framework inspired by the worst-case lower-bound instance. For the makespan objective, we give an FPTAS for the offline variant and a $(2+\epsilon)$-competitive algorithm for the oblivious variant. Our algorithms for the oblivious variants under both objectives run in time $\mathcal{O}(poly(n/\epsilon))$. Lastly, we show that our results are essentially optimal by providing matching lower bounds.

**2012 ACM Subject Classification** Theory of computation → Scheduling algorithms

**Keywords and phrases** scheduling, total completion time, makespan, LP rounding, competitive analysis, approximation algorithm, NP hardness, PTAS

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.38

**Related Version** *Full Version*: https://arxiv.org/abs/2306.15597

**Acknowledgements** We thank the anonymous reviewers for their many insightful comments and suggestions. Chenyang Xu was supported in part by Science and Technology Innovation 2030 –"The Next Generation of Artificial Intelligence" Major Project No.2018AAA0100900, and the Dean's Fund of Shanghai Key Laboratory of Trustworthy Computing, East China Normal University. Ruilong Zhang was supported by NSF grant CCF-1844890.

[1] This work was partially done when the author was a student at City University of Hong Kong.
* All authors (ordered alphabetically) have equal contributions and are corresponding authors.




# 1 Introduction

With increased interest in applying scheduling algorithms to solve real-life problems, many models and methods have been addressing the uncertainty in the scheduling community. Several elegant models that capture uncertainty have been studied in the past two decades, most of which fall under the umbrella of the research on robust optimization [24, 7, 23, 29] or stochastic optimization [16, 18, 15, 14]. In those settings, the uncertainty is usually described by the input. In robust optimization, the input consists of several scenarios, while the input is sampled from a known distribution in stochastic optimization. In some practical cases, we can gain additional information about the input by paying extra costs, e.g., money, time, energy, memory, etc. This model is also known as *explorable uncertainty*, which aims to study the trade-offs between the exploration cost and the quality of a solution.

An intriguing scheduling model for explorable uncertainty was proposed by Dürr et al. [10] under the name of *scheduling with testing*. In their model, before executing a job, one can invest some time to *test* that job, potentially reducing its processing time. A practical use case is code optimization, where we could either simply run programs/codes (jobs) as they are or preprocess them through a code optimizer to hopefully improve their execution times.

Their model considers the test cost as the time spent by the machine, which is certainly important and captures many applications stated in [10]. However, it may fail to describe some scenarios. For example, in the code optimization problem, the code optimizer may be an expert who might need to be employed by other companies. This situation is usually faced by cloud computing companies [25, 6], which accept some tasks and want to assign them to servers. They can employ experts to optimize some time-intensive tasks to speed up the execution. In this way, the server can finish more tasks, thus creating more profit for the company. After optimizing, the experts return the optimized tasks to the cloud computing company, and the company can start to assign tasks to servers. Thus, optimizing does not use servers' time. Different tasks may require a different amount of effort from the expert to optimize and therefore needs different cost. The company has a fixed budget and aims to select some tasks to optimize (test) such that the total processing time of tasks is minimized.

Informally, we consider a natural variant of the model proposed in [10], in which we are given a set of $n$ jobs and a total budget $B$ for testing. Each job $j$ has an upper limit on the processing time $p_j^\wedge$ and testing cost $c_j$. After testing, the processing time of job $j$ decreases to a lower limit $p_j^\vee$, which is possibly hidden for the algorithms. We refer to the model as the *offline* version if $p_j^\vee$ is known by the algorithm for all jobs $j$; otherwise, it is called *oblivious* version. The paper considers two objectives: the total completion time objective and the makespan objective, which are two well-studied objectives for scheduling problems in the literature [19, 5, 22, 9]. The formal definition of our problem is stated in Section 2.

Note that the offline version of the model in Dürr et al. [10] is easy, even if testing a job $j$ requires a job-specific amount of time $t_j$. Testing a job is then beneficial if $p_j^\wedge - p_j^\vee > t_j$. In contrast, we show that the offline version of our budgeted variant of the problem is NP-hard, assuming each job takes a job-specific amount of budget to be tested. We study both the offline and the oblivious settings. Further, we differentiate between the uniform cost variant, where each job takes one unit of budget to be tested, and a non-uniform variant, where the testing cost is job-specific.

## 1.1 Our Contributions

The paper studies the problem of Scheduling with a Limited Testing Budget (SLTB) under both the total completion time minimization objective ($\text{SLTB}_{\text{TC}}$) and the makespan minimization objective ($\text{SLTB}_{\text{M}}$). For both objectives, we further distinguish the offline and oblivious settings.

Our main results are summarized in Table 1. For the objective of total completion time minimization, in the offline setting, we show that the problem is NP-hard even when all the lower processing times are 0 by a reduction from the PARTITION problem, and then give a PTAS. The PTAS is derived based on a novel LP rounding scheme. Further, we find that there exists an FPTAS if all the jobs share the same lower processing time. For the oblivious setting, we give a $(4+\epsilon)$-competitive deterministic algorithm for any $\epsilon$ (we use the concept of the competitive ratio following the previous work [10]). The ratio is almost tight since we prove that no deterministic algorithm has a competitive ratio strictly better than 4. For the objective of makespan minimization, the main results are derived based on a connection between our problem and the classical 0-1 knapsack problem. We prove that the offline setting is NP-hard and admits an FPTAS, while for the oblivious setting, an almost tight competitive ratio of $2+\epsilon$ can be obtained.

**Table 1** The summary of our results. The vector $\mathbf{p}^\vee := (p_1^\vee, \dots, p_n^\vee)$ is the lower processing time vector, and $\mathbf{p}^\vee \in \mathbb{R}_{\geq 0} \cdot \mathbf{1}$ means that all the entries of the vector share the same value. $\epsilon$ is an arbitrary positive parameter.

| | | UB ($\text{SLTB}_{\text{TC}}$) | LB ($\text{SLTB}_{\text{TC}}$) | UB ($\text{SLTB}_{\text{M}}$) | LB ($\text{SLTB}_{\text{M}}$) |
|---|---|---|---|---|---|
| Offline | $\mathbf{p}^\vee \in \mathbb{R}^n_{\geq 0}$ | PTAS (Thm. 4) | NP-C (Thm. 1) | FPTAS | NP-C |
| | $\mathbf{p}^\vee \in \mathbb{R}_{\geq 0} \cdot \mathbf{1}$ | FPTAS | | | |
| Oblivious | — | $4+\epsilon$ | 4 | $2+\epsilon$ | 2 |

**Paper Organization.** We first state some useful notation in Section 2, and then give an overview of our techniques in Section 3. In the remaining part of the main body (Section 4), we describe a PTAS for the offline SLTB with the total completion time objective, the most interesting and technical part of our work. Due to space limitations, the proofs are omitted in this version. All proofs and our other results can be found in the full version [8].

## 1.2 Related Work

**Explorable Uncertainty.** Scheduling with testing falls under the umbrella of the research on optimization with explorable uncertainty, where some additional information can be obtained through queries. The model under the stochastic setting can be traced back to Weitzman's Pandora's Box problem [28] and it remains an active research area up to the present [17, 11]. The model under the adversarial setting was first coined by Kahan [21] to study the number of queries necessary to obtain an element set's median value. So far, many optimization problems have been considered in this setting, e.g. caching [27], geometric tasks [4], minimum spanning tree [20, 26], knapsack [12] and so on.

**Scheduling with Testing.** The problem of scheduling with testing was first coined by Dürr et al. [10]. They consider a model where each testing operation requires one unit of time and mainly investigate non-preemptive schedules on a single machine to minimize the total completion time or makespan. Since the offline version of the problem (algorithms know the lower processing time of each job) is trivial, they mainly consider the online version. They present a 2-competitive deterministic algorithm for total completion time minimization while the deterministic lower bound is 1.8546. They also gave a 1.7453-competitive randomized algorithm while the randomized lower bound is 1.6257. For makespan minimization, they give a 1.618-competitive deterministic algorithm and show that it is optimal for the deterministic setting. They also present a 4/3-competitive randomized algorithm and show that it is optimal.

Later, Albers and Eckl [2] consider the non-uniform testing case where the testing time depends on the job. They investigate the single-machine preemptive and non-preemptive scheduling to minimize the total completion time or makespan. The offline version of this problem is still trivial, so they mainly consider the oblivious version. They present a 4-competitive deterministic algorithm for total completion time minimization and a 3.3794-competitive randomized algorithm. If preemption is allowed, the deterministic ratio can be further improved to 3.2361. All lower bounds are the same as in the uniform testing case. For makespan minimization, they extend the algorithm proposed in [10] and show that the approximation can be preserved in the non-uniform testing case.

Scheduling with testing on identical machines is also considered in the literature [3]. The authors mainly consider the makespan minimization in both non-preemptive and preemptive settings. They look into the non-uniform testing case. For the preemptive setting, they present a 2 competitive algorithm which is essentially optimal. For the non-preemptive setting, they give a 3.1016-competitive algorithm for the general testing case, and the ratio can be improved to 3 when each test requires one unit of time. Later, Gong et al. [13] improved the non-preemptive ratios to 2.9513 and 2.8081 for non-uniform and uniform testing cases, respectively.

## 2 Preliminaries

An *instance* to Scheduling with a Limited Testing Budget (SLTB) is a 5-tuple $\mathcal{I} = (J, \mathbf{p}^\wedge, \mathbf{p}^\vee, \mathbf{c}, B)$. $J = [n]$ denotes a *set of $n$ jobs*. Each job $j$ has an *upper limit on the processing time* $p_j^\wedge \in \mathbb{R}_{\geq 0}$, a *lower processing time* $p_j^\vee \in [0, p_j^\wedge]$ and a *testing cost* $c_j \in \mathbb{R}_{\geq 0}$. These parameters are collected in the *lower and upper limit processing time vectors* $\mathbf{p}^\vee$ and $\mathbf{p}^\wedge$, respectively, and a *vector of testing costs* $\mathbf{c}$. Additionally, a total amount of *budget* $B \in \mathbb{R}_{\geq 0}$ is given.

Each job $j$ can be executed either in a tested or untested state. When job $j$ is tested, $j$ will take $p_j^\vee$ time to process; otherwise, it requires $p_j^\wedge$ time. If a job is tested, it consumes $c_j$ budget; otherwise, no budget is consumed.

We consider offline and oblivious versions. For the offline version, the algorithm knows the complete instance $\mathcal{I}$. For the oblivious version, the lower processing time vector $\mathbf{p}^\vee$ is hidden from the algorithm, and the remaining information of the instance is known a priori.

In this work, we only consider non-preemptive and, w.l.o.g., gapless schedules on a single machine. Once a job starts executing, other jobs cannot be processed until the current job is finished. Thus, a schedule corresponds to a specific ordering of jobs. We define $I \coloneqq [n]$ to be the *set of positions*. The job in position $i \in I$ will be the $i^{th}$ job executed in the schedule.

A *schedule* $S = (\sigma, J_\vee)$ for an instance $\mathcal{I} = (J, \mathbf{p}^\wedge, \mathbf{p}^\vee, \mathbf{c}, B)$ is defined by a *job order* $\sigma$ and a *testing job set* $J_\vee \subseteq J$. The job order $\sigma : J \to I$ is a bijective function that describes the order in which the jobs are processed (i.e., job $j$ is the $\sigma(j)$-th processed job in the non-preemptive schedule). The testing job set $J_\vee \subseteq J$ represents a set of jobs to test with $\sum_{j \in J_\vee} c_j \leq B$.

Given a schedule $S$, we can indicate whether a job is tested using a *set of types* $T \coloneqq \{\vee, \wedge\}$. We say that $j$ is *of type* $\vee$,$\wedge$ if it is *tested*, *untested*, respectively. If $S$ schedules a job $j$ of type $t$ into position $i$, we also say that *position $i$ is of type $t$*. For a schedule $S = (\sigma, J_\vee)$ and a job $j$, let the *type* $t_S(j)$ *of* $j$ *in* $S$ be $\vee$ if $j \in J_\vee$ and $\wedge$ otherwise. Denote by $C_j \coloneqq \sum_{j' \in J, \sigma(j') \leq \sigma(j)} p_{j'}^{t_S(j')}$ the *completion time* of job $j$ in schedule $S$. The total completion time is the sum of all completion times, i.e., $\sum_{j \in J} C_j$, and the *makespan* is the maximum completion time among all jobs, i.e., $\max_{j \in J}\{C_j\}$.

Given a testing job set $J_\vee$, the optimal ordering of the jobs is easy to determine. The ordering is relevant for the total completion time minimization but not for the makespan minimization. It is a well-known fact that the SPT rule (shortest processing time first) orders the jobs optimally for total completion time minimization. The processing times are in our case $p_j^\vee$ if job $j$ is tested and $p_j^\wedge$ otherwise. Thus, an optimal schedule can be easily constructed from an optimal testing job set $J_\vee$.

# 3 Overview of Techniques

In this section, we focus on the total completion time minimization and give technical overviews for the offline model and the oblivious model.

## 3.1 Offline SLTB under Total Completion Time Minimization

For offline $\text{SLTB}_{\text{TC}}$, we mainly show the following theorem. The NP-hardness is proved via a reduction from the Partition problem. Due to space limitations, the proofs are omitted and can be found in the full version [8], we focus on introducing the high-level ideas of our PTAS, the most interesting and technical part of this paper.

▶ **Theorem 1.** *The offline $SLTB_{TC}$ problem is NP-hard even when the lower processing time of each job is* 0*, and admits a PTAS.*

Our algorithm is based on an integer linear programming (ILP) formulation for offline $\text{SLTB}_{\text{TC}}$. The ILP contains variables $x_{j,i,t}$ that dictate whether job $j \in J$ should be scheduled in position $i \in I$ of type $t \in T$. (See Section 4.1 for the exact definition of this ILP.) The ILP is conceptually similar to the classical matching ILP on bipartite graphs [1], with jobs and positions representing the two disjoint independent sets of the bipartition. A matching would then describe an assignment of jobs to positions. However, there are two main differences. First, we have two variables per pair of job and position (distinguished by the type $t \in T$). This translates to each job-position pair having two edges that connect them in the (multi-)graph. Second, the total cost of jobs tested is restricted by some budget $B$. This causes a dependency when selecting edges in the graph.

Our approach combines a rounding scheme of the ILP with an exploitation of the cost structure of the problem. We relax the ILP to an LP by allowing the variables $x_{j,i,t}$ to take on fractional values between 0 and 1. We start with an optimal LP solution and then continue with our rounding scheme, which consists of two phases. In the first phase, we round the solution such that all fractional variables correspond to the edges of a single cycle in the graph mentioned above. These variables are hard to round directly without overusing the budget. Here we start the second rounding phase. We relax some of the constraints in the LP to be able to continue the rounding process. Specifically, we allow certain positions to schedule two jobs (we call these positions *crowded*). We end up with an integral (but invalid) solution that has some crowded positions. Then, we "decrowd" these positions by moving their jobs to nearby positions (shifting the position of some other jobs one up), and show that we can bound the cost of moving a job this way in terms of its current contribution to the overall cost. Observing that moving a job from position $i$ to position $i'$ (note that positions are counted from right to left) increases that job's contribution by a factor of $i'/i$, if a crowded position lies far to the right ($i$ is small), we cannot afford to move one of its jobs too far away. For example, in the extreme case that the rightmost position is crowded (i.e., $i = 1$), even the smallest possible move of one of its jobs to the second-rightmost position (i.e., $i' = 2$) already doubles that job's contribution. Thus, our algorithm tries to avoid producing crowded positions that lie too far to the right (at small positions).

To this end, the rounding process in this phase is specifically tailored to control where crowded positions can appear in the integral solution. We look at the $f(\epsilon) = 2/\epsilon + 1$ smallest (rightmost) positions that appear on the current path (representing fractional variables), and select one of them (let's call it the cut-position) to cut the path into two halves. This is done such that each half contains $1/\epsilon$ many of the smallest positions on the current path. By shifting workload along each of these two halves, we can make one of them integral. This integral half gives us $1/\epsilon$ positions that are not crowded, while the cut-position might have become crowded (as might any future cut-position in the remaining fractional path). Because we cut somewhere in the $f(\epsilon)$ rightmost positions of the path, we can show that for each crowded position, there are many positions further to the right of the schedule that are not crowded (this is basically what our charging argument formalizes). In the end, this allows us to prove that no job is moved too far from its original position (relative to its original position), keeping the cost increase due to such moves small.

## 3.2 Oblivious SLTB under Total Completion Time Minimization

For the oblivious model where the lower processing time vector $\mathbf{p}^\vee$ is hidden, we show that $(4+\epsilon)$ approximation can be obtained, and further, prove that the ratio is the best possible.

▶ **Theorem 2.** *For oblivious SLTB$_{TC}$ and any $\epsilon > 0$, there exists a deterministic algorithm with a competitive ratio of $(4+\epsilon)$, while no deterministic algorithm can obtain a competitive ratio strictly smaller than 4.*

We start by considering the oblivious uniform $\mathrm{SLTB}_{\mathrm{TC}}$ problem to build some intuition. The uniform case limits the number of tested jobs, i.e., we can test at most $k$ jobs. Clearly, for the worst-case analysis, we can assume that each job $j$ tested by our algorithm has $p_j^\vee = p_j^\wedge$; that is, we exhaust the budget, but no job's processing time gets reduced. In contrast, for all the jobs tested by an optimal solution, their processing times can be reduced to 0. Thus, from this perspective, regardless of which jobs we test, our total completion time remains unchanged, but the optimum depends on our tested jobs because the adversary can only let the job $j$ that is not tested by our algorithm have $p_j^\vee = 0$.

Then we find that the oblivious uniform $\mathrm{SLTB}_{\mathrm{TC}}$ problem is essentially equivalent to the following optimization problem: given a set of jobs $J$ with $\mathbf{p}^\wedge$ and $\mathbf{p}^\vee = \mathbf{0}$, the goal is to select $k$ jobs such that the minimum total completion time obtained by testing at most $k$ unselected jobs is maximized. The selected jobs can be viewed as the jobs tested by our algorithm, while the minimum total completion time obtained by testing unselected jobs is the optimum of oblivious uniform $\mathrm{SLTB}_{\mathrm{TC}}$. When our objective value is fixed, a larger optimum implies a better competitive ratio. For this much easier problem, it is easy to see that the best strategy is selecting the $k$ jobs with the largest upper processing time, which is the set of jobs that would be tested by an optimal solution of $\mathrm{SLTB}_{\mathrm{TC}}$ instance $\mathcal{I} = (J, \mathbf{p}^\wedge, \mathbf{p}^\vee = \mathbf{0}, \mathbf{c} = \mathbf{1}, k)$.

We build on the above argument to give the algorithm for the non-uniform case $\mathcal{I} = (J, \mathbf{p}^\wedge, \mathbf{p}^\vee, \mathbf{c}, B)$. The basic idea is constructing an auxiliary instance $\widetilde{\mathcal{I}} := (J, \mathbf{p}^\wedge, \widetilde{\mathbf{p}^\vee} = \mathbf{0}, \mathbf{c}, B)$, solving the instance optimally or approximately, and returning the obtained solution. Use $\mathrm{ALG}(\cdot)$ and $\mathrm{OPT}(\cdot)$ to denote the objective values obtained by our algorithm and an optimal solution of an input instance, respectively. By the theorem proved in the offline model, we have $\mathrm{ALG}(\widetilde{\mathcal{I}}) \le (1+\epsilon)\mathrm{OPT}(\widetilde{\mathcal{I}})$ for any $\epsilon > 0$. In the analysis, we show that our objective value can be split into two parts: $\mathrm{ALG}(\mathcal{I}) \le 2\mathrm{ALG}(\widetilde{\mathcal{I}}) + 2\mathrm{OPT}(\mathcal{I})$, and therefore, due to $\mathrm{OPT}(\widetilde{\mathcal{I}}) \le \mathrm{OPT}(\mathcal{I})$, a competitive ratio of $(4+2\epsilon)$ can be proved.

The lower bound is shown by a hard instance $\mathcal{I} = (J, \mathbf{p}^\wedge = \mathbf{1}, \mathbf{p}^\vee, \mathbf{c} = \mathbf{1}, B = \frac{n}{2})$, where the adversary always lets our tested jobs have lower processing time 1 and the processing time of any other job be 0. Apparently, any deterministic algorithm's objective value is $n(n+1)/2$, while an optimal solution can achieve a total completion time of $n(n+2)/8$, which implies a lower bound of 4. [2]

## 3.3 $\text{SLTB}_\text{M}$ under Makespan Minimization

▶ **Theorem 3.** *The offline $SLTB_M$ problem is NP-hard and admits an FPTAS, while for oblivious $SLTB_M$, an almost tight competitive ratio of $2+\epsilon$ can be obtained (for any $\epsilon > 0$).*

The offline SLTB problem under makespan minimization is closely related to the classical 0-1 knapsack problem. The classical 0-1 knapsack problem aims to select a subset of items such that (i) the total weight of the selected items does not exceed a given capacity; (ii) the total value of the selected items is maximized. To see the connection, consider the testing cost of each job as the weight of each item and the profit of testing a job $(p_j^\wedge - p_j^\vee)$ as the value of an item. Then we build on the algorithmic idea of the knapsack dynamic programming and design an FPTAS for the offline setting.

We use the same framework as the total completion time minimization model for the oblivious setting and obtain a $(2+\epsilon)$-competitive algorithm. The ratio becomes better here since, for the makespan objective, we have $\text{ALG}(\mathcal{I}) \leq \text{ALG}(\widetilde{\mathcal{I}}) + \text{OPT}(\mathcal{I})$, saving a factor of 2. The lower bound proof is also based on the same hard instance $\mathcal{I} = (J, \mathbf{p}^\wedge = \mathbf{1}, \mathbf{p}^\vee, \mathbf{c} = \mathbf{1}, B = n/2)$. Any deterministic algorithm's makespan is $n$ while the optimum is $n/2$, giving a lower bound of 2.

# 4 Offline Setting for SLTB under Total Completion Time Minimization

This section considers the Scheduling with a Limited Testing Budget problem under total completion time minimization ($\text{SLTB}_\text{TC}$) in the offline setting and aims to show the following theorem.

▶ **Theorem 4.** *There exists a PTAS for $SLTB_{TC}$.*

For convenience, we refer to a problem instance as a pair $\mathcal{I} = (J, B)$, dropping the processing time and cost vectors $\mathbf{p}^\vee$, $\mathbf{p}^\wedge$, and $\mathbf{c}$ (which we assume to be implicitly given). Moreover, in this section, we consider the job positions $I = [n]$ *in reverse order* to simplify the calculations. That is, a job $j$ scheduled in position $i \in I$ is processed as the *$i$-th last* job.

## 4.1 ILP Formulation and Fixations

We start by introducing our ILP formulation of the $\text{SLTB}_\text{TC}$ problem and defining the term *fixation* of a (relaxed) instance of our ILP. Such fixations allow us to formally fix the values of certain variables in the (relaxed) ILP when analyzing our algorithm.

[2] Since in the worst-case, the upper and lower processing times of jobs tested by the algorithm are equal, it does not help if the algorithm can be adaptive, i.e., change its testing strategy based on such an information.

**ILP Formulation.** Our ILP has indicator variables $x_{j,i,t}$ that are 1 if job $j$ is scheduled at position $i$ of type $t$ and 0 otherwise. The *contribution* of such a job to the total completion time is[3] $i \cdot p_j^t$. We have constraints to ensure that each of the $n$ positions schedules one job, that each job is scheduled once, and that the cost of tested jobs do not exceed the budget. The equivalence between ILP solutions and $\text{SLTB}_{\text{TC}}$ schedules is formalized in Lemma 5.

Consider an instance $\mathcal{I} = (J, B)$ of the $\text{SLTB}_{\text{TC}}$ problem. We define an ILP $ILP_{\mathcal{I}}$, with the variables $x_{j,i,t}$ for each job $j \in J$, position $i \in I$ and type $t \in T$.

$$\min \sum_{j \in J, i \in I, t \in T} i \cdot p_j^t \cdot x_{j,i,t}$$

$$\text{s.t.} \sum_{j \in J, t \in T} x_{j,i,t} = 1 \quad \forall i \in I \quad (1) \qquad \sum_{i \in I, t \in T} x_{j,i,t} = 1 \quad \forall j \in J \quad (2)$$

$$\sum_{j \in J, i \in I} c_j x_{j,i,\vee} \leq B \quad (3) \qquad x_{j,i,t} \in \{0, 1\} \ \forall j \in J, i \in I, t \in T \quad (4)$$

For $ILP_{\mathcal{I}}$ with variable set $X_{\mathcal{I}} \coloneqq \{ x_{j,i,t} \mid j \in J, i \in I, t \in T \}$, a *solution* $x : X_{\mathcal{I}} \to \mathbb{R}$ assigns each variable in $X_{\mathcal{I}}$ a value. Solution $x$ is called *valid* if it satisfies the four constraints and *invalid* otherwise. For a (possibly invalid) solution $x$ for $\mathcal{I}$ we define its *cost* as $C_{\mathcal{I}}(x) \coloneqq \sum_{j \in J, i \in I, t \in T} i \cdot p_j^t \cdot x_{j,i,t}$ and its *budget use* $B_{\mathcal{I}}(x) \coloneqq \sum_{j \in J, i \in I} c_j x_{j,i,\vee}$ (we omit $\mathcal{I}$ from $C_{\mathcal{I}}$ and $B$ if it is clear from the context). We refer to the different constraints as (1) *position constraints*, (2) *job constraints*, (3) *budget constraint*, and (4) *integrality constraints*.

▶ **Lemma 5.** *Let $\mathcal{I} = (J, B)$ be an instance for $SLTB_{TC}$. For each valid solution $x$ to $ILP_{\mathcal{I}}$ there exists a schedule $S$ for $\mathcal{I}$ with $C(S) = C(x)$ and vice versa. Each can be computed from the other in polynomial time.*

**Relaxation and Fixations.** Our algorithm and analysis use relaxed variants of $ILP_{\mathcal{I}}$ that *fix* certain ILP variables (indicating that, e.g., certain jobs must be tested). It also keeps track of *crowded* positions, in which our algorithm may (temporarily) schedule two jobs (violating the position constraints). We introduce the notion of a *fixation* $\mathcal{F}$ to formally define these relaxed variants $LP_{\mathcal{I},\mathcal{F}}$ of $ILP_{\mathcal{I}}$. [4]

▶ **Definition 6.** A *fixation* $\mathcal{F} = (J(\mathcal{F}), X(\mathcal{F}), I^C(\mathcal{F}))$ of an $\text{SLTB}_{\text{TC}}$ instance $\mathcal{I}$ consists of:

1. a set of *tested* jobs $J(\mathcal{F}) \subseteq J$,
2. a set of *fully-fixed* variables $X(\mathcal{F}) \subseteq X_{\mathcal{I}}$ where $j \notin J(\mathcal{F})$ for all $x_{j,i,t} \in X(\mathcal{F})$, and
3. a set of *crowded positions* $I^C(\mathcal{F}) \subseteq I$.

For a set operator $\circ \in \{\cup, \cap, \setminus\}$ and a set of positions $\bar{I} \subseteq I$, we use the notation $\mathcal{F} \circ \bar{I} \coloneqq (J(\mathcal{F}), X(\mathcal{F}), I^C(\mathcal{F}) \circ \bar{I})$ to express the change to the crowded positions of $\mathcal{F}$.

Given a fixation $\mathcal{F}$, we define the following relaxed variant $LP_{\mathcal{I},\mathcal{F}}$ of $ILP_{\mathcal{I}}$:

1. For each $x_{j,i,t} \in X_{\mathcal{I}}$ we relax the integrality constraint to $0 \leq x_{j,i,t} \leq 1$ (*unit constraints*).
2. For each $x_{j,i,t} \in X(\mathcal{F})$, we add the constraint $x_{j,i,t} = 1$ (*fully-fixed constraints*).
3. For each $j \in J(\mathcal{F})$, we add the constraint $\sum_{i \in I} x_{j,i,\vee} = 1$ (*tested job constraints*).
4. For each $i \in I^C(\mathcal{F})$, we relax the position constraint to $\sum_{j \in J, t \in T} x_{j,i,t} \in \{0, 1, 2\}$.

The resulting LP is omitted in this version and can be found in the full version [8].

[3] Remember that we consider the position in *reverse* order. Thus, the job at position $i$ is the $i$-th last job.

[4] Our PTAS will enumerate through a polynomial number of fixations, and solve the problem for each one of them. The approximation guarantee is then derived for the fixation that is consistent with the optimal solution.

## 4.2 Graph-theoretic Perspective & Paths

Consider an $\text{SLTB}_{\text{TC}}$ instance $\mathcal{I} = (J, B)$ with fixation $\mathcal{F}$ and a (fractional) solution $x$ to $LP_{\mathcal{I},\mathcal{F}}$. The main building block of our algorithm is a rounding scheme based on the following graph-interpretation of $\mathcal{I}$ and corresponding paths based on the current solution $x$:

▶ **Definition 7.** The *instance graph* $G_{\mathcal{I}} \coloneqq (J \cup I, E)$ is a bipartite multi-graph between the jobs $J$ and positions $I$ with exactly two edges between any pair $j \in J$ and $i \in I$. We identify the edge set $E$ with the variable set $X_{\mathcal{I}}$ and refer to a variable $x_{j,i,t} \in E = X_{\mathcal{I}}$ also as an *edge of type* $t \in \{\vee, \wedge\}$ *between* $j$ *and* $i$.

▶ **Definition 8.** A *path* $P$ *in solution* $x$ is a weighted path from $i_s \in I$ (*start position*) to $i_e \in I$ (*end position*) in $G_{\mathcal{I}}$, where the weight of an edge $x_{j,i,t} \in P$ is its value in $x$. $P$ is called *integral* if all its weights are integral and *fractional* if they are all (strictly) fractional.

Nodes and edges in $P$ must be pairwise distinct, except for possibly equal start and end positions $i_s = i_e$, in which case we refer to $P$ also as a *cycle*. We define $J(P)$ as the path's set of jobs, $I(P)$ as its set of positions, and $K(P) \coloneqq I(P) \setminus \{i_s, i_e\}$. Moreover, $X(P)$ is the sequence of edges/variables from start to end position in $P$. We say the $i$-th edge in $X(P)$ is even/odd if $i$ is even/odd, such that $P$ reaches $j \in J(P)$ via an odd edge $x_j^O$ and leaves $j$ via an even edge $x_j^E$. We similarly use $t_j^O$ and $t_j^E$ to denote the type of $x_j^O$ and $x_j^E$, respectively.

Next, we define *shift operations*, which move workload along paths by increasing the volume of one job at any position $i \in I(P)$ while decreasing the volume of another job at $i$.

▶ **Definition 9.** A $\delta$*-shift* of a path $P$ in $x$ decreases the value of all odd edges (variables) of $P$ by $\delta$ and increases the value of all even edges (variables) of $P$ by $\delta$.

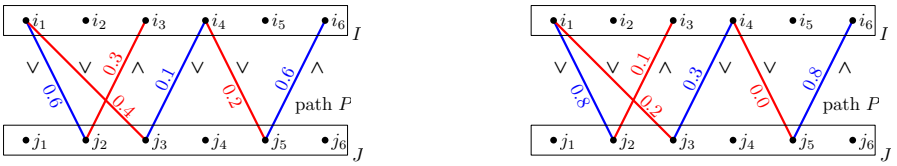
Shift operations (see Figure 1) change the budget use $B(x)$ at a path-dependent (positive or negative) *budget rate* (defined below) and might create crowded positions. Our algorithm's first two phases (Sections 4.3 and 4.4) carefully pair shift operations such that performing paired shifts does not increase the budget and does not create too many crowded positions.

Define the *budget rate* of a path $P$ to be $\Delta(P) \coloneqq \sum_{j \in J(P)} c_j \cdot (\mathbb{1}_{|t_j^E = \vee} - \mathbb{1}_{|t_j^O = \vee})$. Let $P$ be a path in a solution $x$ for $LP_{\mathcal{I},\mathcal{F}}$ without crowded positions (i.e., $I^C(\mathcal{F}) = \varnothing$). $P$ is called $y$*-alternating* (or simply *alternating*) if all odd edges have weight $y$ and all even edges have weight $1 - y$. Lemma 10 below formalizes the effect of a $\delta$-shift in terms of the path's budget rate. Since our analysis can be restricted to paths with very specific, alternating edge values, we also formalize such *alternating paths* and show how they are affected by $\delta$-shifts (see also Figure 2).

▶ **Lemma 10.** *Let $P$ be a path in a solution $x$ for $LP_{\mathcal{I},\mathcal{F}}$ without crowded positions (i.e., $I^C(\mathcal{F}) = \varnothing$). Shifting $P$ in $x$ by $\delta$ yields a (possibly invalid) solution $\tilde{x}$ with $B(\tilde{x}) = B(x) - \delta \cdot \Delta(P)$. If $P$ is $y$-alternating in $x$, then it is $(y - \delta)$-alternating in $\tilde{x}$.*

## 4.3 First Phase: Eliminating all but one cycle

Consider an optimal valid solution $x$ to $LP_{\mathcal{I},\mathcal{F}}$ without crowded positions (i.e., $I^C(\mathcal{F}) = \varnothing$). Lemma 11 below is our main tool for rounding fractional variables in $x$. Consider a set of variables that form a fractional path in $x$. Essentially, we want to use a shift operation from Definition 9 on such a path to make some of its variables integral. If such a shift increases the budget use $B(x)$ (rendering the solution invalid), we can suitably shift a second path (possibly using a negative $\delta$) in parallel to ensure that the budget use $B(x)$ does not increase.




**(a)** Path $P$ with start and end positions $i_3$ and $i_6$. **(b)** Same path $P$ after the shift.

**Figure 1** A path $P$ in a solution $x$ before and after a shift by $\delta = 0.2$. Edges are labeled with their type ($\vee$ or $\wedge$) and weight from a given solution $x$. Odd edges are red, and even edges are blue. The budget rate computes as $\Delta(P) = c_{j_2}(1-0) + c_{j_3}(1-1) + c_{j_5}(0-1) = c_{j_2} - c_{j_5}$.


**(a)** Alternating paths $P$ (solid) and $P'$ (dashed). **(b)** Same paths by $\delta = 0.3$ and $\delta' = 0.6$, respectively.

**Figure 2** Illustration of Lemma 11 for alternating paths $P = (i_3, \dots, i_4)$ (solid) and $P' = (i_4, \dots, i_6)$ (dashed) with budget rates $\Delta(P) = c_{j_3} \coloneqq 2$ and $\Delta(P') = c_{j_5} \coloneqq 1$. Shifting $P$ by $\delta = 0.3$ and $P'$ by $\delta' = 0.6$ keeps the budget use constant. $P$ and $P'$ stay alternating, $P'$ becomes integral, and $i_3, i_4, i_6$ (the start/end positions) violate the position constraints after these shifts.

Such shifts might also cause the violation of the position constraints at the path's start and end positions. We keep track of such violations by adding those positions to the crowded position set $I^C(\mathcal{F})$ of the fixation $\mathcal{F}$. Lemma 11 formalizes this approach (see also Figure 2).

▶ **Lemma 11.** *Consider $x$ a valid solution for $LP_{\mathcal{I},\mathcal{F}}$. Let $P$ be a fractional path in $x$ with $\Delta(P) = 0$ or $P, P'$ be two fractional paths in $x$ with $X(P) \neq X(P')$ and $\Delta(P), \Delta(P') \neq 0$. We can efficiently shift $P$ (and $P'$, if existing) in $x$ to yield a valid solution $\tilde{x}$ for $LP_{\mathcal{I},\tilde{\mathcal{F}}}$ with:*

1. *$C(\tilde{x}) \leq C(x)$ and $B(\tilde{x}) = B(x)$*
2. *$\tilde{\mathcal{F}} = \mathcal{F} \cup I'$, where $I'$ is the set of all start and end positions of non-cyclic paths involved.*
3. *$\tilde{x}$ contains more integral variables than $x$.*

Lemma 11 allows us to shift along general paths (instead of cycles) at the cost of creating crowded positions. We rely on this in Section 4.4 and deal with the crowded positions in Section 4.5. However, to keep the number of crowded positions small and reduce their impact on the final solution, we apply Lemma 11 on cycles for as long as possible. This avoids the creation of crowded positions since shifts along cycles cannot change the net workload at any position. Indeed, note that given any node in a path $P$ that is incident to a fractional edge must have a second fractional edge, or it would violate its job/position constraint. This allows us to complete any fractional path to a cycle. Thus, we can keep applying Lemma 11 to *cycles* (not creating crowded positions) until there is at most one cycle with a non-zero budget rate left (a *blocking cycle*). Let $x$ be a solution to $LP_{\mathcal{I},\mathcal{F}}$. A path $P$ of $x$ is called *critical* if $X(P) = \{ x_{j,i,t} \in X_{\mathcal{I}} \mid x_{j,i,t} \in (0,1) \}$. A *blocking cycle* of $x$ is a critical cycle $P$ with $\Delta(P) \neq 0$. Lemma 12 formalizes the idea above.

▶ **Lemma 12.** *Let $\mathcal{I}$ be an instance, $\mathcal{F}$ be a fixation with $I^C(\mathcal{F}) = \varnothing$, and $x$ be a valid optimal solution to $LP_{\mathcal{I},\mathcal{F}}$. Then we can compute in polynomial time a valid optimal solution $\tilde{x}$ to $LP_{\mathcal{I},\mathcal{F}}$ such that all variables in $\tilde{x}$ are integral, or we find a blocking cycle of $\tilde{x}$. Further, if $P$ is a blocking cycle of $x$, then $P$ is alternating.*

## 4.4 Second Phase: Rounding the blocking cycle

Assume that we used Lemma 12 to compute a blocking cycle $P$ for a solution $x$ to $LP_{\mathcal{I},\mathcal{F}}$ with $I^C(\mathcal{F}) = \varnothing$. Because $P$ is critical, all fractional variables are in $X(P)$. Also, since $\Delta(P) \neq 0$, applying Lemma 11 directly is impossible. Instead, we cut up $P$ repeatedly into two paths $P_1, P_2$, and then use Lemma 11 on these paths (see Algorithm 1). Cutting is done by selecting any position $i \in K(P)$, and separating the path at $i$: $P_1$ will be the path starting at the start position of $P$ and end at $i$. $P_2$ will be the path starting at $i$ and ending at the end position of $P$. We abbreviate this operation by $P_1, P_2 \leftarrow \textsc{Cut}(P, i)$. The drawback of this approach is that Lemma 11 does not guarantee that the resulting solutions still fulfill the position constraints of the start/end positions of $P_1, P_2$, respectively. That is why we add them to $I^C(\mathcal{F})$ in the process.

Algorithm RepeatedCut starts with a solution $\tilde{x}$ and a critical path $\tilde{P}$. It cuts $\tilde{P}$ at some position $i \in K(\tilde{P})$ that is selected by a procedure SelectCutPosition (which is described later in Algorithm 2 in the next subsection). The algorithm then applies Lemma 11 to the two resulting paths, making at least one of them integral (as guaranteed by Lemma 10). After that, $\tilde{x}$ and $\tilde{P}$ are updated accordingly. RepeatedCut finishes when $|K(\tilde{P})| = 0$ (and therefore $\tilde{P}$ cannot be cut into two paths anymore). In such a case, RepeatedCut will reschedule that job to obtain an integral solution. It is also possible that no path remains after the application of Lemma 11. For such a case, $\tilde{x}$ is already integral. Thus, in both cases, the resulting integral solution $\tilde{x}$ is returned.

**Algorithm 1** RepeatedCut.

**Input:** A valid solution $x$ for $LP_{\mathcal{I},\mathcal{F}}$ with $I^C(\mathcal{F}) = \varnothing$, a blocking cycle $P$ of $x$.

$\tilde{P}, \tilde{x} \leftarrow P, x$
**while** $\Delta(\tilde{P}) \neq 0$ **do**
  **if** $K(\tilde{P}) = \varnothing$ **then**
    $j \leftarrow$ unique job in $J(\tilde{P})$; $i_1, i_2 \leftarrow$ remaining two positions in $I(\tilde{P})$
    In $\tilde{x}$, reschedule $j$ into position $\min(i_1, i_2)$ of type $\vee$ if $t_j^O = t_j^E = \vee$ and $\wedge$ else
    **return** $\tilde{x}$
  $i \leftarrow \textsc{SelectCutPosition}(\tilde{P})$
  $P_1, P_2 \leftarrow \textsc{Cut}(P, i)$
  Apply Lemma 11 to $P_1, P_2$ in $\tilde{x}$, changing $\tilde{x}$ accordingly
  **if** both paths became integral **then return** $\tilde{x}$
  $\tilde{P} \leftarrow$ the remaining fractional path
Apply Lemma 11 to $\tilde{P}$ in $\tilde{x}$, changing $\tilde{x}$ accordingly
**return** $\tilde{x}$

In the following, we make statements about the state of the variables involved in the execution of RepeatedCut at the beginning of an iteration of its **while**-loop. Consider the state of RepeatedCut (called on path $P$) at the beginning of the $l$'th iteration of the **while**-loop ($l \geq 1$). We denote by $I_l^C$ the start/end position of $P$ together with all positions selected by SelectCutPosition so far, and $I_*^C$ the start/end position of $P$ together with all positions selected by SelectCutPosition throughout the algorithm. Similarly, denote by $\tilde{x}_l$,$\tilde{P}_l$ the values of $\tilde{x}$,$\tilde{P}$ at that point, respectively, and $\tilde{x}_*$ for the returned solution by RepeatedCut. Denote $\tilde{\mathcal{F}}_l := (J(\mathcal{F}), X(\mathcal{F}), I_l^C)$ and $\tilde{\mathcal{F}}_* := (J(\mathcal{F}), X(\mathcal{F}), I_*^C)$.

▶ **Lemma 13.** *The following is a loop invariant of RepeatedCut for iteration $l \geq 1$: $\tilde{x}_l$ is a valid solution for $LP_{\mathcal{I},\tilde{\mathcal{F}}_l}$ and $\tilde{P}_l$ is a critical fractional alternating path in $\tilde{x}_l$, of which the start and end positions are in $I_l^C$. Also, $\tilde{x}_*$ is an integral valid solution for $LP_{\mathcal{I},\tilde{\mathcal{F}}_*}$.*

Based on Lemma 13, we can analyze the objective obtained by RepeatedCut:

▶ **Lemma 14.** *Consider an application of* RepeatedCut *on solution $x$ for $LP_{\mathcal{I},\mathcal{F}}$ and a blocking cycle $P$. It returns in polynomial time a solution $\tilde{x}$ with $C(\tilde{x}) \leq C(x) + Z$, where $Z$ is either 0 or the contribution of job $j$ rescheduled by* RepeatedCut *in line 5 and $j \notin J(\tilde{\mathcal{F}}_*)$.*

## 4.5 Third Phase: Dealing with crowded positions

Lemma 14 guarantees that applying the algorithm RepeatedCut will return us an integral solution. However, that solution is valid for $LP_{\mathcal{I},\mathcal{F}}$ where $I^C(\mathcal{F})$ still contains some positions. Some of these positions may schedule two jobs, which makes this schedule not valid for $ILP_{\mathcal{I}}$. Our general strategy in this subsection is to move the jobs such that the cost of the solution does not increase too much. In Observation 15, we move each job to a new position and bound the cost created by that operation.

▶ Observation 15. Let $x$ be an integral solution to $LP_{\mathcal{I},\mathcal{F}}$ for some fixation $\mathcal{F}$. Consider a job $j \in J$ that is scheduled in position $i \in I$ of type $t \in T$. Then rescheduling $j$ into position $i'$, i.e., setting $x_{j,i,t} \leftarrow 0$ and $x_{j,i',t} \leftarrow 1$ produces a (possibly invalid) solution $\tilde{x}$, in which the contribution of $j$ increases by a factor of $i'/i$ compared to $x$.

As mentioned above, there are still some positions that schedule two jobs. To obtain an integral valid solution for $ILP_{\mathcal{I}}$, we have to move the jobs in the schedule to new positions, such that there is exactly one job per position scheduled. We want to use Observation 15 to bound the increase in contribution for each job moved this way. Generally, we move the jobs as follows: For a position $i$ that schedules two jobs $j, j'$, we (arbitrarily) distribute $j, j'$ among positions $i, i+1$, thereby moving all jobs from positions $i+1, \ldots, n$ to one higher position. Following this strategy, jobs in higher positions may get moved multiple times.

To bound the contribution in terms of Observation 15, we set up a charging scheme: Each position with two jobs scheduled should be charged to a distinct set of $1/\epsilon$ smaller positions that schedule one job, where $\epsilon$ is the accuracy parameter of our algorithm ($1/\epsilon \in \mathbb{N}$). In the following, we will always use the following function SelectCutPosition for RepeatedCut:

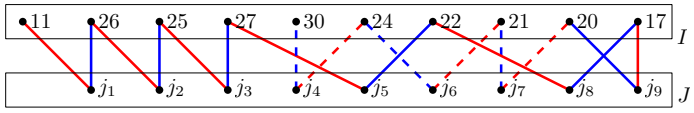
**Algorithm 2** SelectCutPosition.

**Input:** A path $\tilde{P}$ with $|K(\tilde{P})| \geq 1$
**Output:** A position $i \in K(\tilde{P})$
1: **if** $|K(\tilde{P})| \geq 2/\epsilon + 1$ **then**
2: $I' \leftarrow$ the smallest $2/\epsilon + 1$ positions in $K(\tilde{P})$
3: **return** the position that appears as $(1/\epsilon + 1)$-st position in $\tilde{P}$ of the positions in $I'$
4: **else**
5: **return** any position in $K(\tilde{P})$

We care about two properties of the position selected by SelectCutPosition. First, when we cut $\tilde{P}$ into $P_1, P_2$ in line 8 of RepeatedCut, $K(P_1)$,$K(P_2)$ should each contain at least $1/\epsilon$ positions. This way, whichever of these paths becomes integral, there will be $1/\epsilon$ positions that will never be selected by SelectCutPosition in the future. This is important for our charging scheme to have enough positions to charge to. Second, we specifically care about the selected positions being the smallest positions that appear in $K(\tilde{P})$. This essentially allows us to charge each position with two jobs scheduled exclusively to smaller positions, independent of which of the two paths becomes integral.

We represent the charging scheme using a *charging function* (formally defined in Definition 16). Essentially, for a set of positions $\bar{I} \subseteq I$, it charges each position in $\bar{I}$ to its distinct $1/\epsilon$ many smaller positions.


**(a)** Illustration of $\mu_l$ (e.g., $\mu_l(24) = \{16, 23\}$). **(b)** Illustration of $\mu_{l+1}$ (e.g., $\mu_{l+1}(26) = \{16, 23\}$).




**(c)** Path $\tilde{P}_l$ (solid *and* dashed), and path $\tilde{P}_{l+1}$ (solid) after one iteration of the **while**-loop in RepeatedCut.

**Figure 3** An update step of Lemma 17 for $\epsilon = 1/2$. The inner positions of $\tilde{P}_l$ are $K(\tilde{P}_l) = \{17, 20, 21, 22, 24, 25, 26, 27\}$. Its $2/\epsilon + 1 = 5$ smallest positions appear in order $22, 17, 20, 21, 24$. $\tilde{P}_l$ is cut at the $(1/\epsilon + 1) = 3$-rd of these positions (20) into two paths, which are then shifted such that the dashed one becomes integral and the solid one becomes $\tilde{P}_{l+1}$. $I^+_{l+1}$ loses positions 21 and 24 compared to $I^+_l$, as these positions belonged to the dashed path, which became integral. $I^+_{l+1}$ now consists of all remaining $2/\epsilon + 1 = 5$ inner positions $K(\tilde{P}_{l+1}) = \{17, 22, 25, 26, 27\}$. We set $\mu_{l+1}(25) = \mu_l(21)$, $\mu_{l+1}(26) = \mu_l(24)$, and $\mu_{l+1}(27) = \{21, 24\}$ (the lost positions from $I^+_l$).

▶ **Definition 16.** Let $x$ be a solution to $LP_{\mathcal{I},\mathcal{F}}$ for an instance $\mathcal{I}$ and a fixation $\mathcal{F}$. Let $P$ be a critical path in $x$. For a set $\bar{I} \subseteq I$, a *charging function* for $\bar{I}$ is a function $\mu : \bar{I} \to \mathcal{P}(I \setminus \bar{I})$ such that for all $i \in \bar{I}$: (1) $|\mu(i)| = 1/\epsilon$, (2) $\forall i' \in \mu(i) : i' < i$ and (3) $\forall i' \in \bar{I} : \mu(i) \cap \mu(i') = \varnothing$.

Consider the $l$'th iteration of the **while**-loop in RepeatedCut. We define the *charging set* $I^+_l := \bar{I} \cup I^C_l$, where $\bar{I}$ contains the smallest $2/\epsilon + 1$ positions in $K(\tilde{P}_l)$ (or all of them, if $|K(\tilde{P}_l)| < 2/\epsilon + 1$). Similarly, define $I^+_* := I^C_*$.

Lemma 17 shows how to obtain a charging function $\mu_* : I^+_* \to \mathcal{P}(I \setminus I^+_*)$ from a charging function $\mu_1 : I^+_1 \to \mathcal{P}(I \setminus I^+_1)$. We do this by updating the charging function with every iteration of RepeatedCut's loop. Figure 3 exemplifies the update of the charging function.

▶ **Lemma 17.** *If there exists a charging function $\mu_1 : I^+_1 \to \mathcal{P}(I \setminus I^+_1)$, then there also exists a charging function $\mu_* : I^+_* \to \mathcal{P}(I \setminus I^+_*)$.*

We now use Observation 15 together with a charging function (of which we assume the existence for now) on a solution $x$ for $LP_{\mathcal{I},\mathcal{F}}$ produced by RepeatedCut to find a solution for $ILP_{\mathcal{I}}$ with not too much more cost. Lemma 18 will allow us to produce such a solution.

▶ **Lemma 18.** *Let $x$ be a solution returned by RepeatedCut for $LP_{\mathcal{I},\mathcal{F}}$, and let $\mu_*$ be a charging function for $I^+_*$. Then we can find a valid solution $\tilde{x}$ in polynomial time for $ILP_{\mathcal{I}}$ such that the contribution of each job increases by a factor of at most $(1 + \epsilon)$ compared to $x$.*

Consider a solution $x$ returned by RepeatedCut for $LP_{\mathcal{I},\mathcal{F}}$. To be able to apply Lemma 18 and find a solution for $ILP_{\mathcal{I}}$, we need to make sure that we can find a charging function $\mu_1$ for $I^+_1$. Furthermore, we still need to bound the contribution created by the job $j$ in line 5 of RepeatedCut as of Lemma 14. To do this, we choose a proper fixation $\mathcal{F}^*$ and show that a charging function can then be derived.

▶ **Definition 19.** Let $x^*$ be an optimal solution to $ILP_{\mathcal{I}}$ for an instance $\mathcal{I}$. For $i \in I$, let further $j_i \in J$ and $t_i \in T$ such that $x^*_{j_i,i,t_i} = 1$. Using $M := (2/\epsilon + 1)/\epsilon$, we define the fixation $\mathcal{F}^*$ by

$$X(\mathcal{F}^*) = \{ x_{j_i,i,t_i} \mid i \in [M] \} \qquad J(\mathcal{F}^*) = \{ j \in J \mid p^\wedge_j > \min_{i \in [M]} p^{t_i}_{j_i} \} \qquad I^C(\mathcal{F}^*) = \varnothing$$

▶ **Lemma 20.** *Let $x$ be a solution returned by RepeatedCut for $LP_{\mathcal{I},\mathcal{F}^*}$. Then there exists a charging function $\mu_1$ for $I^+_1$.*

Essentially, we brute-force which jobs will be scheduled in the last few positions. This will make sure that these positions are not in $I_1^+$, and as such can be used for the charging function $\mu_1$. Assuming that we brute-forced correctly, an optimal solution will also test all jobs with a larger upper processing time than any of the brute-forced jobs. This is because an optimal solution will always schedule in order of increasing processing times. Finally, we can piece together all of the above lemmas and prove the main theorem (Theorem 4). The detailed proof can be found in [8].

## 5 Conclusion

We initiated the study of Scheduling with a Limited Testing Budget, where we have a limited budget for testing jobs to potentially decrease their processing time. We provided NP-hardness results, a PTAS, as well as tight bounds for a semi-online (oblivious) setting.

Our results open promising avenues for future research. For the setting where we minimize the total completion time, it remains open whether NP-hardness holds for uniform testing cost. Also, while our LP-rounding-based PTAS achieves the best possible approximation, it remains open whether there is a faster, combinatorial algorithm. Another natural direction would be to consider the case of multiple machines.

Another exciting direction is the following *bipartite matching with testing* problem that generalizes our problem, arising from the graph-theoretic perspective in Section 4.2: Consider a bipartite graph $G \coloneqq (L \cup R, E)$ in which each edge $e \in E$ has a cost $c_e$ that can be reduced to $\check{c}_e$ via a testing operation. Given the possibility to test edges before adding them to the matching, we seek a min-cost perfect matching that respects a given testing budget.

## References


1 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows – Theory, algorithms and applications*. Prentice Hall, 1993.

2 Susanne Albers and Alexander Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *WAOA*, volume 12806 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2020.

3 Susanne Albers and Alexander Eckl. Scheduling with testing on multiple identical parallel machines. In *WADS*, volume 12808 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2021.

4 Richard Bruce, Michael Hoffmann, Danny Krizanc, and Rajeev Raman. Efficient update strategies for geometric computing with uncertainty. *Theory Comput. Syst.*, 38(4):411–423, 2005.

5 J. Bruno, E.G. Coffman Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Comm. ACM*, 17:382–387, 1974.

6 João Manuel Paiva Cardoso, José Gabriel de Figueired Coutinho, and Pedro C Diniz. *Embedded computing for high performance: Efficient mapping of computations using customization, code transformations and compilation*. Morgan Kaufmann, 2017.

7 Qingyun Chen, Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Min-max submodular ranking for multiple agents. *CoRR*, abs/2212.07682, 2022. `arXiv:2212.07682`.

8 Christoph Damerius, Peter Kling, Minming Li, Chenyang Xu, and Ruilong Zhang. Scheduling with a limited testing budget, 2023. `arXiv:2306.15597`.

9 J. Du and J.Y.-T. Leung. Complexity of scheduling parallel task systems. *SIAM J. Discrete Math.*, 2(4):473–487, 1989.

10 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. An adversarial model for scheduling with testing. *Algorithmica*, 82(12):3630–3675, 2020.

11 Evangelia Gergatsouli and Christos Tzamos. Online learning for min sum set cover and pandora's box. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 7382–7403. PMLR, 2022.

12 Marc Goerigk, Manoj Gupta, Jonas Ide, Anita Schöbel, and Sandeep Sen. The robust knapsack problem with queries. *Comput. Oper. Res.*, 55:12–22, 2015.

13 Mingyang Gong, Randy Goebel, Guohui Lin, and Eiji Miyano. Improved approximation algorithms for non-preemptive multiprocessor scheduling with testing. *Journal of Combinatorial Optimization*, 44(1):877–893, 2022.

14 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. *Math. Oper. Res.*, 46(1):115–133, 2021.

15 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic makespan minimization in structured set systems. *Math. Program.*, 192(1):597–630, 2022.

16 Anupam Gupta, Benjamin Moseley, and Rudy Zhou. Minimizing completion times for stochastic jobs via batched free times. *CoRR*, abs/2208.13696, 2022. `arXiv:2208.13696`.

17 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *IPCO*, volume 7801 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2013.

18 Varun Gupta, Benjamin Moseley, Marc Uetz, and Qiaomin Xie. Greed works - online algorithms for unrelated machine stochastic scheduling. *Math. Oper. Res.*, 45(2):497–516, 2020.

19 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513—-544, 1997.

20 Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihalák, and Rajeev Raman. Computing minimum spanning trees with uncertainty. In *STACS*, volume 1 of *LIPIcs*, pages 277–288. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2008.

21 Simon Kahan. A model for data in motion. In *STOC*, pages 267–277. ACM, 1991.

22 R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum, 1972.

23 Adam Kasperski and Pawel Zielinski. On the approximability of robust spanning tree problems. *Theor. Comput. Sci.*, 412(4-5):365–374, 2011.

24 Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143, 2016.

25 Parul Kudtarkar, Todd F DeLuca, Vincent A Fusaro, Peter J Tonellato, and Dennis P Wall. Cost-effective cloud computing: a case study using the comparative genomics tool, roundup. *Evolutionary Bioinformatics*, 6:EBO–S6259, 2010.

26 Nicole Megow, Julie Meißner, and Martin Skutella. Randomization helps computing a minimum spanning tree under uncertainty. *SIAM J. Comput.*, 46(4):1217–1240, 2017.

27 Chris Olston and Jennifer Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *VLDB*, pages 144–155. Morgan Kaufmann, 2000.

28 Martin Weitzman. *Optimal search for the best alternative*, volume 78(8). Department of Energy, 1978.

29 Gang Yu and Panagiotis Kouvelis. Complexity results for a class of min-max problems with robust optimization applications. In *Complexity in numerical optimization*, pages 501–511. World Scientific, 1993.

# A $(3/2+\varepsilon)$-Approximation for Multiple TSP with a Variable Number of Depots

**Max Deppert**
Institute for Algorithms and Complexity, Hamburg University of Technology, Germany

**Matthias Kaul**
Institute for Algorithms and Complexity, Hamburg University of Technology, Germany

**Matthias Mnich**
Institute for Algorithms and Complexity, Hamburg University of Technology, Germany

## Abstract

One of the most studied extensions of the famous Traveling Salesperson Problem (TSP) is the Multiple TSP: a set of $m \geq 1$ salespersons collectively traverses a set of $n$ cities by $m$ non-trivial tours, to minimize the total length of their tours. This problem can also be considered to be a variant of Uncapacitated Vehicle Routing, where the objective is to minimize the sum of all tour lengths. When all $m$ tours start from and end at a *single* common *depot* $v_0$, then the metric Multiple TSP can be approximated equally well as the standard metric TSP, as shown by Frieze (1983).

The metric Multiple TSP becomes significantly harder to approximate when there is a *set* $D$ of $d \geq 1$ depots that form the starting and end points of the $m$ tours. For this case, only a $(2-1/d)$-approximation in polynomial time is known, as well as a 3/2-approximation for *constant* $d$ which requires a prohibitive run time of $n^{\Theta(d)}$ (Xu and Rodrigues, *INFORMS J. Comput.*, 2015). A recent work of Traub, Vygen and Zenklusen (STOC 2020) gives another approximation algorithm for metric Multiple TSP with run time $n^{\Theta(d)}$, which reduces the problem to approximating metric TSP.

In this paper we overcome the $n^{\Theta(d)}$ time barrier: we give the first efficient approximation algorithm for Multiple TSP with a *variable* number $d$ of depots that yields a better-than-2 approximation. Our algorithm runs in time $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$, and produces a $(3/2+\varepsilon)$-approximation with constant probability. For the graphic case, we obtain a deterministic 3/2-approximation in time $2^d \cdot n^{\mathcal{O}(1)}$.

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

**Keywords and phrases** Traveling salesperson problem, rural postperson problem, multiple TSP, vehicle routing

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.39

**Funding** Partially supported by DFG project MN 59/4-1.

**Acknowledgements** The third author thanks László Végh for inspiring discussions on multi-depot TSP and feedback on an earlier version.

## 1 Introduction

The Traveling Salesperson Problem (TSP) is one of the best-studied problems in combinatorial optimization: given a complete graph $G$ on $n$ nodes together with edge weights $w : E(G) \to \mathbb{R}_{\geq 0}$, we seek a tour that starts at some node $v_0 \in V(G)$, then visits all other nodes of $G$ exactly once, and returns to the origin $v_0$ in such a way that the overall tour weight is minimized, which is the sum of the weights of the edges traversed by the tour. TSP is one of Karp's 21 NP-complete problems [15], which motivates the design of efficient, polynomial-time approximation algorithms for it. Recall that an $\alpha$-approximation for a minimization problem returns, for any instance $I$, in polynomial time a solution of value at


most $\alpha \cdot \mathrm{OPT}(I)$, where $\mathrm{OPT}(I)$ denotes the value of an optimal solution for $I$. Of special importance in this regard is Metric TSP, when the edge weight function $w$ obeys the triangle inequality. For Metric TSP, the tree doubling heuristic yields a 2-approximation, which was improved to a 3/2-approximation by Christofides [7] and Serdyukov [24] in the 1970s. This approximation factor stood unchallenged for many decades until its recent improvement to a $(3/2 - 10^{-36})$-approximation by Karlin et al. [14].

Due to its ubiquity, a large variety of extensions of the TSP have been studied. Among the most prominent ones is the Multiple TSP, where a set of $m \geq 1$ salespersons (all starting from some common node $v_0$ called a *depot*) jointly traverse the entire set of $n$ nodes, in order to minimize the overall tour length. That is, the goal is to find a collection of $m$ pairwise edge-disjoint cycles $C_1, \ldots, C_m$ (all intersecting in some node $v_0$) in $G$ whose union covers all nodes of the graph and such that the sum of the weights of the cycles is minimized. This character of having to solve both a partitioning and a sequencing problem simultaneously gives rise to considerable added complexity, akin to that encountered in vehicle routing problems. Indeed, one could interpret this problem as a variant of the Uncapacitated Vehicle Routing Problem; we, however, will adhere to the TSP-style naming convention, since this is more prevalent in the literature. Let us just mention here that for metric edge weights, Multiple TSP has the same approximation guarantee as the standard (single-person) metric TSP; in particular, Metric Multiple TSP admits a $(3/2-10^{-36})$-approximation in polynomial time by the results of Karlin et al. [14]. Frieze [10] analysed the case of Metric Multiple TSP when each of the tours has to contain at least one edge and intersect a common depot $v_0$; he provided a 3/2-approximation for this setting in polynomial time. The Multiple TSP is studied in more than 1,300 publications; an extensive survey is provided by Bektaş [1].

In this paper we study an extension of the Multiple TSP, where a *set* $D \subseteq V(G)$ of nodes is distinguished as *depots*. Formally, the Multi-Depot Multiple TSP (mdmtsp) takes as input a complete graph $G$ on $n$ nodes together with edge weights $w : E(G) \to \mathbb{R}_{\geq 0}$, as well as a set $D \subseteq V(G)$ of $d = |D|$ depots and an integer $m \geq 1$ denoting the number of salespersons available. Now again we are seeking a set of $m$ pairwise edge-disjoint cycles $C_1, \ldots, C_m$ in $G$ whose union covers all nodes of the graph and such that the sum of the weights of the cycles is minimized, but in addition each cycle must contain some depot from $D$. Such set of cycles is an optimal solution for the mdmtsp instance, and we denote the value of some optimal solution by $\mathrm{OPT}(G, D, w)$ (or simply OPT if the instance is clear from the context). The mdmtsp is motivated by several applications of high practical impact, like motion planning of a set of unmanned aerial vehicles [17, 21, 31] and the routing of service technicians where the technicians are leaving from multiple depots [20].

The theoretical aspects of mdmtsp have been studied in many research papers [1, 2, 3, 6, 13, 16, 25, 26, 28, 29, 30]. At this point, let us issue a word of caution. There are quite a few other varieties of (mdm)tsp considered in the literature, all subtly different from each other. For a compact overview of possible variations, there is the review paper of Bektaş [1]. For the scope of this paper, we consider the metric mdmtsp where the edge weights form a metric. This allows us to assume that $m = d$ throughout. This assumption is made for the following two reasons: on the one hand, the case $m > d$ is negligible as the objective function (the total weight of all tours) is invariant for multiple tours starting from a single depot (if weights satisfy the triangle inequality, it is easy to show that there is always an optimal solution in which at most one route will start and end at each depot). On the other hand, in the case $m \leq d$ we can try each selection of $d' = m$ depots by paying a multiplicative factor of $\binom{d}{m}$ in the run time only. Thus, any instance of metric mdmtsp is specified by a triple $(G, D, w)$, where $G$ is a complete graph on $n$ nodes, $D \subseteq V(G)$ is the set of depots, and $w : E(G) \to \mathbb{R}_{\geq 0}$ is a metric.

*The polynomial-time approximability of metric* MDMTSP *is not fully understood.* That there is a *set* $D$ of depots (and not just a single depot $v_0$), each one of which must be visited by one of the tours, makes the approximability of the problem much harder compared to metric MULTIPLE TSP (i.e., the version without depots). The added complexity arises from the fact that we not only have to give a good order in which to visit nodes, as in the TSP, but we also have to partition the nodes appropriately. In particular, the Christofides-Serdyukov algorithm [7, 24] no longer yields a 3/2-approximation in this setting. The original analysis of Christofides' and Serdyukov's algorithms relies on all odd-degree nodes of some spanning structure $F$ lying on the same tour, so a parity-correcting edge set $J$ can be computed that weighs at most $\frac{1}{2}$ OPT. This fact is not available in the multi-depot setting, so in polynomial time we can only guarantee a 2-approximation by using the spanner $F$ for $J$ also. However, this only achieves a tight approximation ratio of $2 - \frac{1}{d}$ for the multi-depot setting, as shown by Xu et al. [30] (see Figure 1 for a version of their lower-bound example), because the matching can have weight $\frac{d-1}{d}$ OPT.

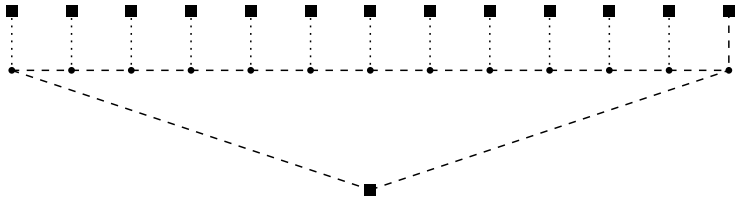


**Figure 1** An instance on which Algorithm 1 achieves approximation ratio arbitrarily close to 2. Square nodes are depots and all edges have unit weight. Dashed edges indicate a tour of length $d$, dotted edges a minimum CSF with weight $d-1$ that requires a join of weight $d-1$ to complete.

To avoid this issue, the constrained spanning forest needs to be rearranged such that there is again a matching of weight $\frac{1}{2}$ OPT, as in the work of Xu and Rodrigues [28] – this rearrangement though requires $n^{\Theta(d)}$ time.

Similarly, the algorithmic approaches to metric TSP based on solving a linear program (LP) are also unlikely to give $\alpha$-approximation algorithms with $\alpha < 2$ for metric MDMTSP. To this end, consider the following multi-depot version of the subtour-elimination LP, MDMTSP-LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E(G)} w_e x_e & & \\ \text{subject to} \quad & \sum_{e \in \delta(v)} x_e = 2, & \forall\, v \in V(G) \setminus D & \\ & \sum_{e \in \delta(U)} x_e \geq 2, & \forall\, U \subseteq V(G) \setminus D & \\ & x_e \in [0,2], & \forall e \in E(G) & \end{aligned} \tag{MDMTSP-LP}$$

In Figure 2 we give a construction to show that MDMTSP-LP has integrality gap 2.

If one gives up on the polynomial run time of the approximation algorithm, then smaller approximation factors are possible. Xu and Rodrigues [28] show how to obtain a 3/2-approximation, but their algorithm requires time $n^{\Theta(d)}$, which is polynomial only if the number $d = |D|$ of depots is constant. Another 3/2-approximation for MDMTSP with run time $n^{\Theta(d)}$ follows from the recent work of Traub, Vygen and Zenklusen [26]. They in fact show the much stronger result that any $\lambda$-approximation algorithm for metric TSP also gives a $(\lambda + \varepsilon)$-approximation algorithm for metric MDMTSP with an additional run time factor of $n^{\mathcal{O}(d/\varepsilon)}$. In summary, the state-of-the-art for metric MDMTSP is that there is no $\alpha$-approximation known for MDMTSP for any absolute constant $\alpha < 2$ which runs in time $n^{o(d)}$.

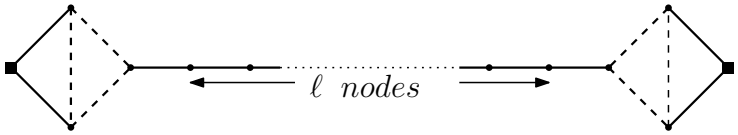



**Figure 2** An instance on which the multi-depot subtour-elimination LP has integrality gap arbitrarily close to 2. The square nodes are the depots and all edges have unit weight. The dashed edges are assigned $x_e = 0.5$ by the LP, the other edges $x_e = 1$. The LP then has optimum value at most $\ell + 6$, whereas the MDMTSP has optimum value $2(\ell + 4)$.

## 1.1 Our Results

Our main result is a novel approximation algorithm for metric MULTIPLE TSP on $d$ depots with a significantly improved run time. That is, we provide the first algorithm for metric MDMTSP which breaks the $n^{\Theta(d)}$ time barrier to obtain an approximation ratio strictly better than 2. Given an instance $(G, D, w)$ of metric MDMTSP, we say a collection $C_1, \ldots, C_d$ of cycles is a *tour* if they jointly cover all nodes of $G$ and each cycle contains exactly one depot from $D$.

▶ **Theorem 1.** *There is an algorithm that, given any $\varepsilon > 0$, in time $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$ computes a tour $T$ for any set of $n$ cities with metric distances and $d$ depots. The algorithm is randomized, and with constant probability the length of the tour $T$ is at most $(3/2+\varepsilon) \cdot \mathrm{OPT}$.*

Thus, our result significantly improves on the previously best run time $n^{\Theta(d)}$ by Xu and Rodrigues [28] at the cost of some small additive $\varepsilon$ in the approximation factor.

To break through the barrier of 2 on the approximation ratio, we need to rework the initial spanner $F$ to be "correctly aligned" with the optimal solution so that each subtour contains an even number of odd-degree nodes, as initially proposed by Xu and Rodrigues [28]. We show that an approximate reworking can be done in time $f(d, \varepsilon) \cdot n^{\mathcal{O}(1)}$ for some suitable function $f$, resulting in a $(3/2+\varepsilon)$-approximation. To this end, firstly, we give a reduction of metric MDMTSP to a related routing problem which is known as the RURAL POSTPERSON PROBLEM (RPP). In the RPP, we are given an edge-weighted graph $G$ and a set $R$ of required edges, and are asked to compute a minimum-weight edge set $F$ such that $R \dot\cup F$ is connected and Eulerian. Our reduction reveals an approximation algorithm with run time $\mathcal{O}(n^3 + t)$ to compute solutions no worse than $\frac{3}{2}\,\mathrm{OPT} + \frac{1}{2}w(T)$, where $w(T)$ is the weight of a single-person TSP tour $T$ through the depots and $t$ denotes the time to compute $T$. Then we use a randomized algorithm of Gutin et al. [11] for the RPP, and an approximate weight reduction scheme of van Bevern et al. [27], to construct a $(1+\varepsilon)$-approximation algorithm for a variant of RPP with depots.

We are then in a position to speed up the reworking of the inital spanner due to two key insights. Firstly, we allow for some misalignment to remain, as long as it is only due to the presence of some *light* edges, limiting the number of edges we have to consider for removal from $F$. Secondly, we employ the constructed approximation algorithm for the RPP to complete our now disconnected spanner to a tour. Doing this provides a large speedup over the algorithm of Xu and Rodrigues, who first need to guess a set of edges to reconnect their spanner, and then employ a matching algorithm to obtain a tour. Using the RPP allows us to do both of these steps simultaneously, and considerably faster.

An important special case of metric TSP is when the metric $w$ is induced by the shortest paths in a graph. This version is also known as GRAPHIC TSP, and has been studied extensively from the perspective of approximation algorithms [18, 23]. For MDMTSP on graphic metrics, we obtain a *deterministic algorithm* with slightly better approximation factor and a reduced run time.

▶ **Theorem 2.** *There is an algorithm that, given any graph $G$ on $n$ nodes and set $D \subseteq V(G)$ of $d$ depots, in time $2^d \cdot n^{\mathcal{O}(1)}$ computes a tour $T$ of length at most $\frac{3}{2} \cdot \mathrm{OPT}$.*

## 1.2 Related work

In the case of a single salesperson, i. e. $m = 1$, Bérczi et al. [4] gave a polynomial-time 3/2-approximation for the *many-visits* version of metric MDMTSP, that is, when each node $v$ is equipped with a request $r(v)$ (encoded in binary) of how many times it should be visited.

In a different work, Bérczi et al. [5] have shown constant-factor approximation algorithms with ratio at most 4 for variants of metric MDMTSP where each tour has to visit exactly one depot (and thus, $d \leq m$).

For the RPP, which asks for a minimum-weight tour traversing all edges of a given subset $R$ of edges of a graph, there is a polynomial-time approximation algorithm (cf. Frederickson [8] or Jansen [12]) similar to the approach of Christofides-Serdyukov for metric TSP. The weight of a solution can be bounded by $(3\,\mathrm{OPT} + w(R))/2$, where $w(R)$ is the weight of $R$.

Oberlin et al. [19] studied heuristic approaches for MDMTSP where $d = m$ and each salesperson is located at its own depot.

For the objective of minimizing the longest tour length of any salesperson (rather than the sum of all the tour lengths), Frederickson et al. [9], among other routing problems, considered the case of a single depot ($d = 1$), and presented a $(\rho + 1 - 1/m)$-approximation algorithm where $\rho$ is the approximation ratio of an algorithm for the single-salesperson TSP.

# 2 Preliminaries

Let $\mathcal{U}$ be a finite universe. For a function $w : \mathcal{U} \to \mathbb{R}$ and a multiset $U \subseteq \mathcal{U}$ we write $w(U)$ to mean $\sum_{u \in U} w(u)$, where the sum has an additional summand for each copy of an element in $U$, i.e. it considers multiplicities. The disjoint union $\dot{\bigcup}_i A_i$ of some sets $\{A_i\}_i$ is considered to be the multiset of all items in the collection. For brevity we often write $2A$ to mean $A \,\dot{\cup}\, A$.

Throughout this paper, we consider the multiple-depot version of the metric MULTIPLE TSP, or metric MDMTSP for short. We will generally represent the metric by an edge-weighted graph whose shortest-path metric we assume to be the metric in use. Notice that this makes no difference in our setting since we are allowed to traverse edges multiple times; only if this is forbidden does the non-metric case become relevant.

▶ **Definition 3.** *An instance $(G, D, w)$ of metric* MDMTSP *consists of a complete graph $G$, a set $D \subseteq V(G)$ of $d$* depots*, and a metric $w : E(G) \to \mathbb{R}_+$ on $V(G)$. A multiset of edges $T \subseteq E(G)$ is called a* tour *of $(G, D, w)$ if*

***(P1)*** *the multigraph $(V(G), T)$ has even degree at every node in $V(G)$,*
***(P2)*** *and each connected component of $(V(G), T)$ contains at least one node from $D$.*

*We denote by $\mathrm{OPT}(G, D, w)$ the minimum weight of any tour of $(G, D, w)$. If the instance is clear from the context, we may only say* OPT*.*

Edge sets are generally allowed to be multisets, and graphs can have parallel edges.

Imitating the general framework of Christofides-Serdyukov [7, 24], we first compute an edge set $F$, called a constrained spanning forest (CSF), that ensures the connectivity property (P2). We then compute an additional set of edges $J$ such that $F \,\dot{\cup}\, J$ has property (P1).

▶ **Definition 4.** *Let $G$ be a graph and let $D \subseteq V(G)$. A* constrained spanning forest *in $(G, D)$ is a set $F \subseteq E(G)$ of edges such that the graph $(V(G), F)$ is acyclic and every connected component of $(V(G), F)$ contains at least one node from $D$.*

We will make use of the following result.

▶ **Theorem 5** (Rathinam et al. [21]). *Given any graph $G$ on $n$ nodes and $m$ edges with weights $w : E(G) \to \mathbb{R}$ and a set $D \subseteq V(G)$, a minimum-weight* CSF *of $(G, D, w)$ can be computed in time $\mathcal{O}((n+m)\log n)$.*

**Proof.** The computation of a minimum-weight CSF for $(G, D, w)$ can be reduced to computing a minimum-weight spanning tree in a graph $G'$ with edge weights $w'$. The graph $G'$ is obtained from $G$ by adding a single root node $r$ connected to every depot by an edge of some weight less than the weight all other edges in $G$. Kruskal's algorithm is guaranteed to choose these edges for the minimum-weight spanning tree in $(G', w')$, and after removing $r$ we are left with a minimum-weight CSF for $(G, D, w)$. ◀

Traditionally, property (P1) is obtained by computing a minimum-weight matching on the odd-degree nodes of the CSF, as in Algorithm 1. However, this algorithm only achieves a

**Algorithm 1** Algorithm Multi-Depot Christofides-Serdyukov.

**Input:** A metric MDMTSP instance $(G, D, w)$.
**Output:** A tour $T$ with $w(T) \leq 2\,\mathrm{OPT}$.
1 Compute a minimum-weight CSF $F$ for $(G, D, w)$;
2 Let $U$ be the set of nodes with odd degree in $F$;
3 Compute a minimum-weight perfect matching $M$ in $G[U]$;
**Result:** $T := F \dot\cup M$

tight approximation ratio of $2 - \frac{1}{d}$ for the multi-depot setting, as shown by Xu et al. [30] (see Figure 1 for a simplified version of their lower bound example), because the matching can have weight $\frac{d-1}{d}\,\mathrm{OPT}$. To avoid needing such an expensive matching, the constrained spanning forest needs to be rearranged such that there is again a matching of weight $\frac{1}{2}\,\mathrm{OPT}$, as in the work of Xu and Rodrigues [28].

## 3 Reducing Multi-Depot Multiple TSP to Rural Postperson Problem

In this section we show a reduction from the metric MDMTSP to the RPP. Recall that in the RPP there is a required set $R$ of edges that a tour should traverse, rather than a set of nodes.

▶ **Definition 6** (Rural Postperson Problem). *An instance $(G, R, w)$ of* RPP *consists of a graph $G$, a set $R \subseteq E(G)$ of required edges, and a metric weight function $w : E(G) \to \mathbb{R}_{\geq 0}$. A* solution *is multiset $J \subseteq E(G)$ for which $(V, R \dot\cup J)$ is Eulerian, and which has only one non-singleton connected component.*[1] *The weight of a solution $J$ is $w(J) = \sum_{e \in J} w(e)$. The goal of* RPP *is to compute an* optimal *solution, which is a solution of minimum weight* $\mathrm{OPT}(G, R, w)$.

There is a polynomial-time approximation algorithm for RPP [8, 12] which computes a solution $J \subseteq E(G)$ such that $w(R \dot\cup J) \leq \frac{3}{2} w(R \dot\cup J^*)$, i. e. $w(J) \leq \frac{3}{2}\,\mathrm{OPT} + \frac{1}{2} w(R)$, where $J^*$ is some optimal solution. Due to the first inequality and the unavoidable weight of $R$, the algorithm is known as a 3/2-approximation for RPP. This situation is very similar to the current approximation status of metric MDMTSP, where we can obtain a 3/2-approximation if we allow for some additional additive term. This observation motivates the following reduction from metric MDMTSP to RPP.

[1] This means that nodes not incident to any edge from $R$ do not need to be visited by the computed tour. For metric cost functions, however, one can always reduce to the case where $R$ spans $G$.

▶ **Observation 7.** *For each instance* $(G, D, w)$ *of* MDMTSP *there is an instance* $(G', R, w')$ *of* RPP *such that any solution to the* RPP *instance can be transformed in polynomial time into a solution to the* MDMTSP *instance of the same weight.*

**Proof.** First, compute any TSP tour $S$ on the depots in $G$, that is, on $G[D]$. Then, for each node $v \in V \setminus D$, introduce a second node $v'$, as well as an edge $e_v = \{v, v'\}$, and set its weight to $w'(e_v) = 0$. For each edge $e \in E(G)$ set $w'(e) = w(e)$, and set $R$ to be the union of $S$ and two copies of each $e_v$. The any solution to the constructed instance of RPP corresponds to an MDMTSP tour for $(G, D, w)$ of the same weight. ◀

Notice that this reduction, together with the 3/2-approximation for RPP, allows us to compute a solution to MDMTSP of weight at most $\frac{3}{2}\,\mathrm{OPT} + \frac{1}{2}w(S)$. In particular, if all depots are pairwise close to each other this is already a better-than-2 approximation.

In Section 5 we will in some sense show a stronger result that there is also a (Turing) reduction from MDMTSP to the special case of RPP where $(V, R)$ has few connected components, which has been shown by Gutin et al. to be tractable [11]:

▶ **Proposition 8** (Gutin et al. [11])**.** *There is a randomized algorithm for* RPP *that for any instance* $(G, R, w)$*, where* $(V(G), R)$ *has* $k$ *connected components and* $w$ *takes only integer values, in time* $2^{\mathcal{O}(k)}(n + \mathrm{OPT}(G, R, w))^{\mathcal{O}(1)}$ *produces a solution. With constant probability, the computed solution is optimal.*

However, as our reduction is only $(1+\varepsilon)$-approximate with respect to the solution qualities, and needs time exponential in $d$ and $\varepsilon$, we need to remove the polynomial dependence on $\mathrm{OPT}(G, R, w)$ in Proposition 8. To this end, we will adapt an approximate weight reduction scheme by van Bevern et al. [27]:

▶ **Lemma 9** (adapted from van Bevern et al. [27, Lemma 2.12])**.** *Let* $(G, R, w)$ *be an instance of* RPP *with integral weighs, let* $\varepsilon > 0$*, and let* $\beta = \max\{w(e) \mid e \in E(G)\}$*. Then in polynomial time we can compute a weight function* $w' : E(G) \to \mathbb{N}_{\geq 0}$ *such that*

- $\max\{w'(e) \mid e \in E(G)\} \leq 2|E(G)|/\varepsilon$*,*
- *and for all* $\alpha \geq 1$*, any solution* $J$ *to* $(G, R, w')$ *with weight* $w'(J) \leq \alpha\,\mathrm{OPT}(G, R, w')$ *also fulfills* $w(J) \leq \alpha\,\mathrm{OPT}(G, R, w) + \varepsilon\beta$*, as long as* $J$ *contains at most two copies of each edge.*

**Proof.** The rounding scheme simply sets $w'(e) := \lfloor w(e) \cdot \frac{2|E(G)|}{\varepsilon \cdot \beta} \rfloor$ for each edge $e$ of $G$. This yields the first condition, by definition. For the second condition, observe that for any $J$ we have

$$\frac{\varepsilon \cdot \beta}{2|E(G)|} w'(J) \leq w(J) \leq \frac{\varepsilon \cdot \beta}{2|E(G)|} w'(J) + \frac{\varepsilon \cdot \beta}{2|E(G)|}|J| \leq \frac{\varepsilon \cdot \beta}{2|E(G)|} w'(J) + \varepsilon\beta \ .$$

Hence, the two weight functions are equivalent up to scaling by a constant and the addition of at most $\varepsilon\beta$. ◀

Notice that the restriction on $J$ having at most two copies of each edge is never a problem: whenever a solution to the RPP has three or more copies of one edge, we can delete two of them to obtain a cheaper solution.

We will combine Proposition 8 and Lemma 9 to obtain an approximation for $k$-component RPP whose run time does not depend on OPT.

▶ **Corollary 10.** *There is a randomized algorithm that, for any $\varepsilon > 0$, in time $2^{\mathcal{O}(k)}(n+\frac{1}{\varepsilon})^{\mathcal{O}(1)}$ computes a solution $J$ for any instance $(G, R, w)$ of* RPP *where $(V(G), R)$ has $k$ connected components. The computed solution $J$ has the property that, with constant probability, $w(J) \leq \mathrm{OPT}(G, R, w) + \varepsilon \max\{w(e) \mid e \in J^* \dot\cup R\}$, where $J^*$ is some optimal solution to the instance.*

**Proof.** We first guess the weight $\beta$ of the most expensive edge in $J^* \dot\cup R$, where $J^*$ is some optimal solution. There are only $|E(G)|$ options, so the guessing generates only polynomial overhead. All edges that are more expensive than $\beta$ can be removed from the instance to get some graph $G'$. Now we apply Lemma 9 to get an instance with weights $w'$ bounded by $2|E(G)|/\varepsilon$ and use the exact algorithm from Proposition 8 to get a solution $J$ to $(G', R, w')$ in time $2^{\mathcal{O}(k)}(n+\frac{1}{\varepsilon})^{\mathcal{O}(1)}$. From Lemma 9 with $\alpha = 1$ we know that

$$w(J) \leq \mathrm{OPT}(G, R, w) + \varepsilon\beta \leq \mathrm{OPT}(G, R, w) + \varepsilon \max\{w(e) \mid e \in J^* \dot\cup R\},$$

which proves the claim. ◀

We will be using this algorithm to complete partial solutions to instances of MDMTSP. We will need only a slight modification that allows for the presence of depots as follows.

▶ **Definition 11** (Depot Rural Postperson Problem)**.** *An instance $(G, D, R, w)$ of the* Depot Rural Postperson Problem *(*DRPP*) consists of an* RPP *instance $(G, R, w)$ and some depots $D \subseteq V(G)$. A* solution *is a multiset $J \subseteq E(G)$ such that $(V, R \dot\cup J)$ is Eulerian and each non-singleton connected component of $(V, R \dot\cup J)$ contains at least one depot. The weight of a solution $J$ is $w(J) = \sum_{e \in J} w(e)$. The goal is to compute an* optimal *solution, which is a solution of minimum weight* $\mathrm{OPT}(G, D, R, w)$.

The depot version DRPP can be reduced to regular RPP quite easily.

▶ **Corollary 12.** *There is a randomized algorithm that, for any instance $(G, D, R, w)$ of* DRPP *where $(V(G), R)$ has $k$ connected components and any $\varepsilon > 0$, in time $2^{\mathcal{O}(k \log k)}(n+\frac{1}{\varepsilon})^{\mathcal{O}(1)}$ computes a solution $J$ such that, with constant probability, $w(J) \leq (1+\varepsilon)\,\mathrm{OPT}(G, D, R, w) + \varepsilon w(R)$.*

**Proof.** Note first that each connected component of $(V(G), R)$ can be assumed to contain at most one depot, so $|D| \leq k$. Some optimum solution $J^*$ induces a partition of the connected components of $(v(G), R)$ where each partition class corresponds to those components connected to some specific depot. There are at most $|D|^k \in 2^{\mathcal{O}(k \log k)}$ possible partitions, so we can try each partition, solve the regular RPP instance on each of the $|D|$ classes of the partition using the algorithm from Corollary 10, and return the best solution we found. ◀

## 4 Intuition for the Algorithm

The algorithm of Xu and Rodrigues [28] executes, at a very high level, the following steps:

1. Compute a minimum-weight constrained spanning forest $F$ for $(G, D, w)$.
2. Guess a set $X$ of at most $|D| - 1$ edges such that they are in $F$ but not in some fixed optimal tour $T$.
3. Discard the guessed edges $X$ from $F$. This leaves at most $2|D|$ connected components in $(V, F \setminus X)$. If we have guessed correctly, every subtour of $T$ now contains an even number of odd-degree nodes. There must exist some edges $A$ from $T$ such that $(F \setminus X) \cup A$ is a CSF for $(G, D)$ with $w((F \setminus X) \cup A) \leq w(T)$. The value $|A|$ is at most $|D|$, so we also guess $A$.

4. Since $A$ contains only edges from $T$, every subtour of $T$ still contains an even number of odd-degree nodes with respect to $(V, (F \setminus X) \cup A)$. If we compute an odd-join $J$ for $(F \setminus X) \cup A$, we have $w(J) \leq \frac{1}{2}\,\mathrm{OPT}$, so return $((F \setminus X) \cup A) \dot\cup J$.

Since the algorithm needs to guess $2|D|$ edges in total (in step 2), it can be implemented in time $n^{\mathcal{O}(d)}$. We modify this guessing step by considering for discarding (in step 3) only very heavy edges, and by sidestepping the guessing of $A$; instead of computing first a connected structure and then a join we do this simultaneously, using the algorithm for RPP. Specifically:

- In step 2, we only consider edges that are very expensive relative to the total weight of the forest $F$. If the targeted edge $e$ is not in this collection, we do not delete it but instead use it as part of the augmenting set $A$, doubling the edge. This also fixes parity, but requires us to relax $w((F \setminus X) \dot\cup A) \leq w(T)$ to $w((F \setminus X) \dot\cup A) \leq (1+\varepsilon)w(T)$. The $\varepsilon$ can be controlled by how expensive relative to $F$ we allow these non-deleted edges to be.
- In step 3, we do not actually guess $A$, we merely use its existence. We instead solve an instance of DRPP with at most $2d$ connected components for which $A \dot\cup J$ is a solution. Using the algorithm from Corollary 12, we can compute a $(1+\varepsilon)$-approximation for the DRPP in time $f(d,\varepsilon) \cdot n^{\mathcal{O}(1)}$. We use the solution $J'$ as a replacement for $A \dot\cup J$ knowing $w(J') \leq (1+\varepsilon)w(A \dot\cup J)$. Combining inequalities for $J$ and $F$ gives:

$$w((F \setminus X) \dot\cup J') \leq (1+\varepsilon)w(((F \setminus X) \dot\cup A) \dot\cup J) \leq (1+\varepsilon)\frac{3}{2}\,\mathrm{OPT} \quad .$$

An illustration of the augmentation scheme can be found in Figure 3.

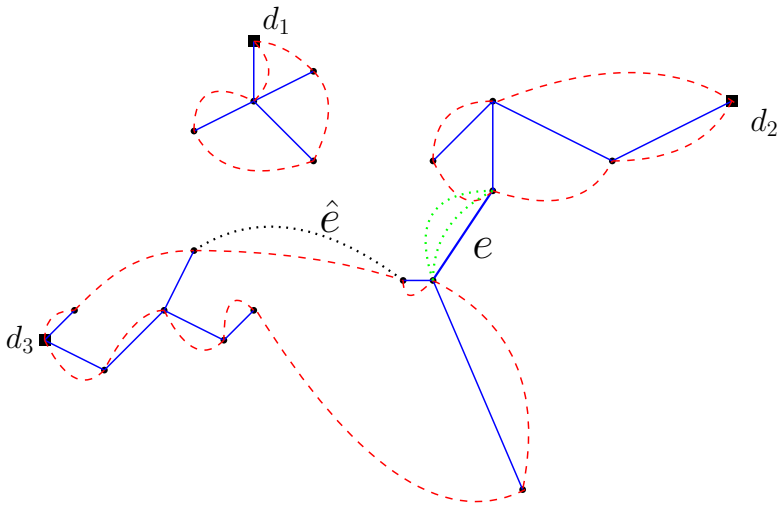



**Figure 3** Illustration of the augmenting edges explored by our algorithm. Blue solid edges represent a CSF, dashed red edges an optimal tour. Note that the tours of $d_2$ and $d_3$ have an odd number of nodes with odd degree in the CSF. Our algorithm considers two options to remedy this. We either add two copies (green, dotted) of the edge $e$ to the optimal tour, if $e$ is considered to be light enough. This joins the tours of $d_2$ and $d_3$ to a single tour with an even number of odd-degree nodes. If $e$ is considered too heavy for this, we remove $e$ from the CSF and replace it with the edge $\hat{e}$ (black, dotted). As $\hat{e}$ comes from the optimal tour, this keeps the cost of the CSF below OPT and it fixes the parities.

## 5 Towards Faster Parity Correction

In this section, we give a formal version of the algorithm described in the previous section, which we state as Algorithm 2. We prove this algorithm to be a $(3/2+\mathcal{O}(\varepsilon))$-approximation for metric MDMTSP in Theorem 17. We will restate and reprove some of the results of Xu and Rodrigues [28] to ensure completeness of the presentation and to integrate properly our changes to their algorithm.

To this end, let us fix some notation throughout this section. Let $(G, D, w)$ be a metric MDMTSP instance with $D = \{d_1, \ldots, d_d\}$, an optimal tour $T$, and the minimum-weight CSF $F$ for $(G, D, w)$ that was computed Algorithm 1. We denote by $T_i$ be the connected component of $T$ containing $d_i$, by $F_i$ the subtree of $F$ containing $d_i$, and $U_i$ the set of nodes in $T_i$ that have odd degree with respect to the edges in $F$. We take $U = \bigcup_i U_i$.

By minimality of $F$, we already know that $w(F) \leq w(T) = \mathrm{OPT}$. To extend $F$ to an MDMTSP tour, we try to compute a minimum-weight matching between the nodes in $U$. It is a standard argument from the analysis of the Christofides-Serdyukov algorithm that if $|U_i|$ is even, $T_i$ contains two disjoint matchings for the nodes in $U_i$. So if every $U_i$ has even cardinality, then any minimum-weight matching has weight at most $\frac{1}{2}\,\mathrm{OPT}$. But this is not the case, since a tree $F_i$ might contain nodes from many different tours, so the odd-degree nodes are distributed arbitrarily. To record this "misalignment" between the trees and subtours we introduce the concept of an *alignment graph*.

▶ **Definition 13** (Alignment Graph). *The alignment graph $H$ for $(G, F, T)$ is constructed as $V(H) = D$, and*

$$E(H) = \{\{d_i, d_j\} \mid \exists e \in F \text{ s.t. } |e \cap V(T_i)| = |e \cap V(T_j)| = 1\} \ .$$

*We also define a weight function $w_H : E(H) \to \mathbb{R}_+$ as*

$$w_H((d_i, d_j)) := \min\{w(e) | e \in F,\ |V(T_i) \cap e| = |V(T_j) \cap e| = 1\} \ .$$

In the following, we assume that $H$ is connected, otherwise the analysis holds independently for each connected component.

Now we take $D_{\mathrm{odd}}$ to be the collection of depots $d_i$ for which $|U_i|$ is odd, and $A_H$ to be any $D_{\mathrm{odd}}$-join in $H$. The join $A_H$ can be used to augment the original tour $T$ to be connected. To do this we transfer the join to the original graph to ensure that it contains a "cheap" matching. For every edge $e = \{d_i, d_j\} \in E(A_H)$, pick an edge in $\hat{e} \in E(F)$ with $w(\hat{e}) = w_H(e)$ and $|\hat{e} \cap V(T_i)| = |\hat{e} \cap V(T_j)| = 1$. Denote by $A$ the collection of these $\hat{e}$. Observe that every node in $T \dot\cup 2A$ has even degree, and every connected component of the graph contains an even number of nodes from $U$. Hence, there exists a $U$-join $J$ in $G$ with $w(J) \leq \frac{1}{2} w(T \dot\cup 2A)$.

Now notice that, if the edges in $A$ have weight at most $\varepsilon w(F)$, this inequality yields that Algorithm 1 already achieves a good approximation ratio, specifically

$$w(F) + w(M) \leq w(F) + w(J) \leq (1+\varepsilon)w(F) + \frac{1}{2}w(T) \leq (\frac{3}{2} + \varepsilon)w(T) \ .$$

Based on this observation, we are willing to augment $T$ with low-weight edges from $F$ to find a low-weight matching. Therefore, we need to distinguish between *heavy* and *light* edges.

▶ **Definition 14.** *Let $\varepsilon > 0$. An edge $e \in F$ is* $\varepsilon$-light *if $w(e) \leq \frac{\varepsilon}{d} \cdot w(F)$; else, it is* $\varepsilon$-heavy.

We now try to replace the $\varepsilon$-heavy edges in $A$ with some other edges from $T$.

**Algorithm 2** Algorithm Extended Multi-Depot Christofides-Serdyukov.

**Input:** A metric MDMTSP instance $(G, D, w)$ and a parameter $\varepsilon > 0$.
**Output:** A tour $T$ such that $w(T) \leq (3/2 + \varepsilon)\,\mathrm{OPT}$ with constant probability.

```
Compute a minimum-weight CSF F for (G, D, w);
Let T be the currently best MDMTSP solution, initially 2F;
Let Y be the set of edges in F which are ε-heavy;
foreach X ⊆ Y, |X| ≤ |D| do
    F' := F \ X;
    Compute a solution M for the DRPP instance (G, D, F', w) using Corollary 12 ;
    if w(M ∪̇ F') < w(T) then
        set T = M ∪̇ F';
    end
end
```

**Result:** $T$

▶ **Lemma 15** (compare [28, Section 2]). *Let $X \subseteq A$. Then there exist a set $\hat{A} \subseteq E(T)$ of edges such that $(F \setminus X) \cup \hat{A}$ is a CSF for $(G, D)$ with $w((F \setminus X) \cup \hat{A}) \leq w(T)$.*

**Proof.** Consider the forest $F'$ obtained from $T$ by removing exactly one edge from each subtour. $F'$ contains only edges from $T$, so it is disjoint from $X$. By a standard matroid exchange argument, for each $e \in X$ there is a $\hat{e} \in F'$ such that $F - e + \hat{e}$ is a CSF and $w(e) \leq w(\hat{e})$. This process can then be iterated to remove all of $X$. The collection of these $\hat{e}$ is $\hat{A}$, giving $w((F \setminus X) \cup \hat{A}) \leq w(F') \leq w(T)$. ◀

This process of replacing augmenting edges from $A$ with edges out of $T$ also fulfills the key goal of putting an even number of odd-degree nodes into every connected component of some augmented MDMTSP solution. Consider the following lemma, which is in substance a version of a statement by Xu and Rodrigues [28, Theorem 2].

▶ **Lemma 16.** *Let $A, X, \hat{A}$ be as in Lemma 15. Then every connected component of $T \cup (A \setminus X)$ contains an even number of nodes that have odd degree in $(F \setminus X) \cup \hat{A}$.*

**Proof.** Notice first that the connected components of $T \cup (A \setminus X)$ are the union of some of the subtours of $T$. Since the edges in $\hat{A}$ belong to some tour, adding them to $F \setminus X$ flips the parity of the degrees of two nodes on the same tour, so the total parity of odd-degree nodes on that tour does not change. We can therefore restrict ourselves to considering the odd-degree nodes with respect to $F \setminus X$.

Recall that originally $A$ was constructed from a $D_{\mathrm{odd}}$-join $A_H$ in the alignment graph $H$. So $X$ corresponds to some edge set $X_H \subseteq A_H$, and we know that $A_H \setminus X_H$ constitutes an $(D_{\mathrm{odd}} \Delta \dot{\bigcup}_{e \in X_H} e)$-join, where $\Delta$ denotes the symmetric difference. For multisets, the symmetric difference of some sets contains an item if and only if it is contained an odd number of times in their disjoint union. At the same time, removing an edge $e \in X$ from $F$ with corresponding $\{d_i, d_j\} \in X_H$ changes the degree of one node in $V(T_i)$ and one node on $V(T_j)$. So, the depots whose tours contain an odd number of odd-degree nodes with respect to $F \setminus X$ are precisely $(D_{\mathrm{odd}} \Delta \dot{\bigcup}_{e \in X_H} e)$, so $A_H \setminus X_H$ joins them correctly. ◀

We are now ready to prove that Algorithm 2 returns a $(3/2 + \mathcal{O}(\varepsilon))$-approximation, with constant probability.

▶ **Theorem 17.** *The tour returned by Algorithm 2 has weight at most $(3/2 + \mathcal{O}(\varepsilon))\,\mathrm{OPT}$, with constant probability.*

**Proof.** Set $T'$ to be the tour returned by the algorithm. Now let $A$ be the augmenting edge set for some optimal tour as before and $X$ the set of $\varepsilon$-heavy edges in $A$. We look at the iteration of the algorithm where that $X$ is considered for removal. From Lemma 15 we know that there exists some edge set $\hat{A}$ with $w(F \setminus X \cup \hat{A}) \leq \mathrm{OPT}$ and Lemma 16 implies that there is an edge set $J$ such that $F \setminus X \dot\cup \hat{A} \dot\cup J$ is Eulerian, contains a depot in each connected component, and $w(J) \leq \frac{1}{2}w(T \cup (A \setminus X))$. Therefore, $\hat{A} \dot\cup J$ is a solution to the DRPP instance $(G, D, F', w)$. Hence, the $M$ computed in the algorithm fulfills $w(M) \leq (1+\varepsilon)w(\hat{A} \dot\cup J) + \varepsilon w(F')$. Putting all these inequalities together yields

$$\begin{aligned} w(T') &= w(M) + w(F \setminus X) \\ &\leq (w(F) - w(X) + w(\hat{A})) + (\frac{1}{2}(w(T) + w(A \setminus X)) + \varepsilon(w(\hat{A}) + w(J) + w(F)) \\ &\leq w(T) + \frac{1}{2}(w(T) + d \cdot \frac{\varepsilon}{d}w(F)) + \varepsilon(w(\hat{A}) + w(J) + w(F)) \leq \frac{3}{2}w(T) + 4\varepsilon w(T), \end{aligned}$$

where we use that $A \setminus X$ contains at most $d$ edges, and all of them are $\varepsilon$-light. ◀

Notice that this algorithm will give a $(3/2+\varepsilon)$-approximation when called with $\varepsilon/4$ as the parameter of approximation. The additional run time cost will vanish in the $\mathcal{O}$-notation. The probability of success for this algorithm is the same as that for the algorithm in Proposition 8. Notice that while that algorithm is called many times, we only need it to succeed for one specific choice of $X$. If it fails in one of the other attempts, we do not care.

It remains to analyze the run time of this algorithm. We see that $Y$, the set of $\varepsilon$-heavy edges, has size at most $\frac{d}{\varepsilon}$, so there are only $(\frac{d}{\varepsilon})^d$ possible values for $X$ to be tried. Note also that each loop iteration requires the approximate solution of a DRPP instance with $\mathcal{O}(d)$ components which can be done in time $2^{\mathcal{O}(d \log d)}(n + \frac{1}{\varepsilon})^{\mathcal{O}(1)}$. The total run time then is $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$, showing Theorem 1.

## 6 A Deterministic 3/2-Approximation for Graphic MDMTSP

In this section we provide a deterministic 3/2-approximation for MDMTSP when the metric is the shortest-path metric of an unweighted graph. The run time of the algorithm is $2^d \cdot n^{\mathcal{O}(1)}$.

Let $(G, D)$ be an instance of graphic MDMTSP, where $G$ this time is the unweighted graph inducing the shortest-path metric. Note that we can assume $G$ to be connected. This allows us to construct TSP tours that are not much more expensive than optimal solutions to MDMTSP, which re-enables the original analysis of the Christofides-Serdyukov Algorithm.

For a given optimal MDMTSP tour $T$, we can extend it to a TSP tour by introducing at most $2(d-1)$ edges. To do this, contract the subtours of $T$, find a spanning tree in the contracted graph, and double all the edges of that tree. We then see that the solution $F \dot\cup M$ returned by Algorithm 1 fulfills

$$w(F \dot\cup M) \leq w(T) + \frac{1}{2}(w(T) + 2(d-1)) = \frac{3}{2}w(T) + d - 1 \ .$$

Notice that the additive term $d-1$ is likely to be very small, since we know $w(T) \geq n - d$. A similar argument can also be made for metrics which are continuous in the sense that the space cannot be partitioned into two very distant parts.

▶ **Observation 18.** *Let $(G, D, w)$ be an* integer-weighted *instance of* MDMTSP *for which there exists a constant $L$ such that, for all $U \subseteq V(G)$, it holds* $\min\{w(u,v) \mid u \in U, v \notin U\} \leq L$. *Then Algorithm 1 returns a solution $T$ for $(G, D, w)$ with $w(T) \leq \frac{3}{2}\,\mathrm{OPT}(G, D, w) + L(d-1)$.*

Since we know $w(T)$ to be in $\Omega(n-d)$ also in this case, Algorithm 1 gives an *asymptotic* 3/2-approximation for any constant $d$ and $L$. We can even get rid of the additive term in the graphic case (i.e. $L = 1$) with some additional run time.

▶ **Observation 19.** *There is a* 3/2*-approximation algorithm for graphic* MDMTSP *with run time* $2^d \cdot n^{\mathcal{O}(1)}$.

**Proof.** Let $T$ be some fixed optimal tour. We start by guessing the set $D' \subseteq D$ of depots whose subtours in $T$ contain at least one edge, generating on overhead of $2^d$. Then we know that $T$ contains a CSF $F'$ for $(G, D')$ with weight $|E(T)| - |D'|$. As before, we connect together all subtours of the depots in $D'$ with $|D'| - 1$ edges, and double these edges. Then the tour $F \dot\cup M$ returned by Algorithm 1 fulfills

$$|E(F \dot\cup M)| \leq |E(T)| - |D'| + \frac{1}{2}(|E(T)| + 2(|D'| - 1)) = \frac{3}{2}|E(T)| - 1,$$

and that proves the claim. ◀

For the special case where we require each depot to have a non-empty tour, we do not even have to guess the correct subset of depots in Observation 19, yielding a 3/2-approximation in truly polynomial time.

## 7 Discussion

We have shown that metric MDMTSP admits a randomized $(3/2+\varepsilon)$-approximation algorithm in time $(1/\varepsilon)^{\mathcal{O}(d \log d)} \cdot n^{\mathcal{O}(1)}$, filling in the gap between the best-known polynomial approximation factor, 2, and the 3/2-approximation of Xu and Rodrigues in time $n^{\Theta(d)}$. However, there remain a number of natural openings for improving on our result:

- Can our algorithm be derandomized? Since we rely on the algorithm of Gutin et al. [11] to solve RPP instances, this would require a derandomization of their result. However, their algorithm relies on the Schwartz-Zippel Lemma [22, 32] for which no deterministic alternatives have been found in the last 40 years.
- Can the approximation factor be improved from $3/2 + \varepsilon$ to 3/2? We loose some approximation quality both when determining which edges to delete from the CSF, and when solving RPP. Improving the first point would require a further refinement of the tree-rearrangement technique introduced by Xu and Rodrigues [28]. For the second point, the RPP algorithm of Gutin et al. would need to be sped up to run in strongly polynomial time. Again, their algorithm relies on algebraic techniques for which derandomization appears difficult, so a major technical innovation for $k$-component RPP is maybe necessary.
- Does there exist some polynomial-time $\alpha$-approximation algorithm for MDMTSP with $\alpha < 2$? We know from Traub et al. [26] that any $\alpha$-approximation algorithm for single-salesperson TSP implies a $(\alpha + \varepsilon)$-approximation for MDMTSP for any constant number of depots, i.e. in time $n^{\Theta(d)}$. For instances with many depots however, the problem remains intractable. It is of particular interest that two major technical tools for the classical TSP, Christofides' Algorithm and the Subtour-Elimination LP, fail to achieve better-than-2-approximations in the multi-depot regime (see Figure 1 and Figure 2). It appears that to make progress on a polynomial-time algorithm some novel structural insights would be required.

## References


1 Tolga Bektaş. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006. doi:10.1016/j.omega.2004.10.004.

2 Tolga Bektaş. Formulations and benders decomposition algorithms for multidepot salesmen problems with load balancing. *European J. Oper. Res.*, 216(1):83–93, 2012. doi:10.1016/j.ejor.2011.07.020.

3 Enrique Benavent and Antonio Martínez. Multi-depot multiple TSP: a polyhedral study and computational results. *Annals Oper. Res.*, 207:7–25, 2013. doi:10.1007/s10479-011-1024-y.

4 Kristóf Bérczi, Matthias Mnich, and Roland Vincze. A 3/2-approximation for the metric many-visits path TSP. *SIAM J. Discrete Math.*, 36(4):2995–3030, 2022. doi:10.1137/22M1483414.

5 Kristóf Bérczi, Matthias Mnich, and Roland Vincze. Approximations for many-visits multiple traveling salesman problems. *Omega*, 116:102816, 2023. doi:10.1016/j.omega.2022.102816.

6 M. Burger, Z. Su, and B. De Schutter. A node current-based 2-index formulation for the fixed-destination multi-depot travelling salesman problem. *European J. Oper. Res.*, 265(2):463–477, 2018. doi:10.1016/j.ejor.2017.07.056.

7 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Carnegie-Mellon University, Management Sciences Research Group, 1976. doi:10.1007/s43069-021-00101-z.

8 Greg N. Frederickson. Approximation algorithms for some postman problems. *J. ACM*, 26(3):538–554, 1979. doi:10.1145/322139.322150.

9 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *SIAM J. Comput.*, 7(2):178–193, 1978. doi:10.1137/0207017.

10 A.M. Frieze. An extension of Christofides heuristic to the $k$-person travelling salesman problem. *Discrete Appl. Math.*, 6(1):79–83, 1983. doi:10.1016/0166-218X(83)90102-6.

11 Gregory Gutin, Magnus Wahlström, and Anders Yeo. Rural postman parameterized by the number of components of required edges. *J. Comput. Syst. Sci.*, 83(1):121–131, 2017. doi:10.1016/j.jcss.2016.06.001.

12 Klaus Jansen. An approximation algorithm for the general routing problem. *Inf. Process. Lett.*, 41(6):333–339, 1992. doi:10.1016/0020-0190(92)90161-N.

13 Imdat Kara and Tolga Bektaş. Integer linear programming formulations of multiple salesman problems and its variations. *European J. Oper. Res.*, 174(3):1449–1458, 2006. doi:10.1016/j.ejor.2005.03.008.

14 Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. In *Proc. STOC 2021*, pages 32–45, 2021. doi:10.1145/3406325.3451009.

15 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum, 1972.

16 Gilbert Laporte, Yves Nobert, and Serge Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transport. Sci.*, 22(3):161–172, 1988. doi:10.1287/trsc.22.3.161.

17 Waqar Malik, Sivakumar Rathinam, and Swaroop Darbha. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Oper. Res. Lett*, 35(6):747–753, 2007. doi:10.1016/j.orl.2007.02.001.

18 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *Proc. FOCS 2011*, pages 560–569, 2011. doi:10.1109/FOCS.2011.56.

19 Paul Oberlin, Sivakumar Rathinam, and Swaroop Darbha. A transformation for a heterogeneous, multiple depot, multiple traveling salesman problem. In *Proc. ACC 2009*, pages 1292–1297, 2009. doi:10.1109/ACC.2009.5160666.

20 SN Parragh. Solving a real-world service technician routing and scheduling problem. In *Proc. TRISTAN VII*, 2010.

21 Sivakumar Rathinam, Raja Sengupta, and Swaroop Darbha. A resource allocation algorithm for multivehicle systems with nonholonomic constraints. *IEEE Trans. Automat. Sci. Engin.*, 4(1):98–104, 2007. doi:10.1109/TASE.2006.872110.

22 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

23 András Sebő and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014. doi:10.1007/s00493-014-2960-3.

24 Anatoliy I. Serdyukov. On some extremal walks in graphs. *Upravlyaemye Sistemy*, 17:76–79, 1978. (in Russian).

25 Kaarthik Sundar and Sivakumar Rathinam. An exact algorithm for a heterogeneous, multiple depot, multiple traveling salesman problem. In *Proc. ICUAS 2015*, pages 366–371, 2015. doi:10.1109/ICUAS.2015.7152311.

26 Vera Traub, Jens Vygen, and Rico Zenklusen. Reducing path TSP to TSP. In *Proc. STOC 2020*, pages 14–27, 2020. doi:10.1137/20M135594X.

27 René van Bevern, Till Fluschnik, and Oxana Yu. Tsidulko. On approximate data reduction for the rural postman problem: Theory and experiments. *Networks*, 76(4):485–508, 2020. doi:10.1002/net.21985.

28 Zhou Xu and Brian Rodrigues. A 3/2-approximation algorithm for the multiple TSP with a fixed number of depots. *INFORMS J. Comput.*, 27(4):636–645, 2015. doi:10.1287/ijoc.2015.0650.

29 Zhou Xu and Brian Rodrigues. An extension of the Christofides heuristic for the generalized multiple depot multiple traveling salesmen problem. *European J. Oper. Res.*, 257(3):735–745, 2017. doi:10.1016/j.ejor.2016.08.054.

30 Zhou Xu, Liang Xu, and Brian Rodrigues. An analysis of the extended Christofides heuristic for the $k$-depot TSP. *Oper. Res. Lett.*, 39(3):218–223, 2011. doi:10.1016/j.orl.2011.03.002.

31 S. Yadlapalli, W.A. Malik, S. Darbha, and M. Pachter. A lagrangian-based algorithm for a multiple depot, multiple traveling salesmen problem. *Nonlinear Analysis: Real World Applications*, 10(4):1990–1999, 2009. doi:10.1016/j.nonrwa.2008.03.014.

32 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Proc. EUROSAM 1979*, volume 72 of *Lecture Notes Comput. Sci.*, pages 216–226, 1979. doi:10.1007/3-540-09519-5_73.

# Efficient Parallel Output-Sensitive Edit Distance

**Xiangyun Ding**
University of California, Riverside, CA, USA

**Xiaojun Dong**
University of California, Riverside, CA, USA

**Yan Gu**
University of California, Riverside, CA, USA

**Youzhe Liu**
University of California, Riverside, CA, USA

**Yihan Sun**
University of California, Riverside, CA, USA

## Abstract

In this paper, we study efficient parallel edit distance algorithms, both in theory and in practice. Given two strings $A[1..n]$ and $B[1..m]$, and a set of operations allowed to edit the strings, the edit distance between $A$ and $B$ is the minimum number of operations required to transform $A$ into $B$. In this paper, we use edit distance to refer to the Levenshtein distance, which allows for unit-cost single-character edits (insertions, deletions, substitutions). Sequentially, a standard Dynamic Programming (DP) algorithm solves edit distance with $\Theta(nm)$ cost. In many real-world applications, the strings to be compared are similar to each other and have small edit distances. To achieve highly practical implementations, we focus on output-sensitive parallel edit-distance algorithms, i.e., to achieve asymptotically better cost bounds than the standard $\Theta(nm)$ algorithm when the edit distance is small. We study four algorithms in the paper, including three algorithms based on Breadth-First Search (BFS), and one algorithm based on Divide-and-Conquer (DaC). Our BFS-based solution is based on the Landau-Vishkin algorithm. We implement three different data structures for the longest common prefix (LCP) queries needed in the algorithm: the classic solution using parallel suffix array, and two hash-based solutions proposed in this paper. Our DaC-based solution is inspired by the output-insensitive solution proposed by Apostolico et al., and we propose a non-trivial adaption to make it output-sensitive. All of the algorithms studied in this paper have good theoretical guarantees, and they achieve different tradeoffs between work (total number of operations), span (longest dependence chain in the computation), and space.

We test and compare our algorithms on both synthetic data and real-world data, including DNA sequences, Wikipedia texts, GitHub repositories, etc. Our BFS-based algorithms outperform the existing parallel edit-distance implementation in ParlayLib in all test cases. On cases with fewer than $10^5$ edits, our algorithm can process input sequences of size $10^9$ in about ten seconds, while ParlayLib can only process sequences of sizes up to $10^6$ in the same amount of time. By comparing our algorithms, we also provide a better understanding of the choice of algorithms for different input patterns. We believe that our paper is the first systematic study in the theory and practice of parallel edit distance.

**2012 ACM Subject Classification** Theory of computation → Parallel algorithms

**Keywords and phrases** Edit Distance, Parallel Algorithms, String Algorithms, Dynamic Programming, Pattern Matching

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.40

**Related Version** *Full Version*: `https://arxiv.org/abs/2306.17461` [16]

**Supplementary Material** *Software (Source Code)*: `github.com/ucrparlay/Edit-Distance` [17]
archived at `swh:1:dir:f5b361f279f694fcde074f24f5e11fb41bddf96e`

**Funding** This work is supported by NSF grants CCF-2103483, CCF-2238358, and IIS-2227669, and UCR Regents Faculty Fellowships.





## 1 Introduction

Given two strings (sequences) $A[1..n]$ and $B[1..m]$ over an alphabet $\Sigma$ and a set of operations allowed to edit the strings, the ***edit distance*** between $A$ and $B$ is the minimum number of operations required to transform $A$ into $B$. WLOG, we assume $m \leq n$. The most commonly used metric is the ***Levenshtein distance*** which allows for unit-cost single-character edits (insertions, deletions, substitutions). In this paper, we use *edit distance* to refer to the Levenshtein distance. We use $k$ to denote the edit distance for strings $A$ and $B$ throughout this paper. Edit distance is usually used to measure the similarity of two strings (a smaller distance means higher similarity).

Edit distance is a fundamental problem in computer science, and is introduced in most algorithm textbooks (e.g., [14, 15, 23]). In practice, it is widely used in version-control software [54], computational biology [12, 31, 39], natural language processing [10, 29], and spell corrections [28]. It is also closely related to other important problems such as longest common subsequence (LCS) [50], longest increasing subsequence (LIS) [34], approximate string matching [56], and multi-sequence alignment [59]. The classic dynamic programming (DP) solution can compute edit distance in $O(nm)$ work (number of operations) between two strings of sizes $n$ and $m$. This complexity is impractical if the input strings are large. One useful observation is that, in real-world applications, the strings to be compared are usually *reasonably similar*, resulting in a relatively small edit distance. For example, in many version-control softwares (e.g., Git), if the two committed versions are similar (within a certain number of edits), the "delta" file is stored to track edits. Otherwise, if the difference is large, the system directly stores the new version. Most of the DNA or genome sequence alignment applications also only focus on when the number of edits is *small* [39]. We say an edit distance algorithm is *output-sensitive* if the work is $o(nm)$ when $k = o(n)$. Many more efficient and/or practical algorithms were proposed in this setting with cost bounds parameterized by $k$ [19, 20, 21, 22, 26, 35, 36, 37, 46, 47, 49].

Considering the ever-growing data size and plateaued single-processor performance, it is crucial to consider parallel solutions for edit distance. Although the problem is simple and well-studied in the sequential setting, we observe a *huge gap* between theory and practice in the parallel setting. The few implementations we know of [7, 55, 58] simply parallelize the $O(nm)$-work sequential algorithm and require $O(n)$ span (longest dependence chain), which indicates low-parallelism and redundant work when $k \ll n$. Meanwhile, numerous theoretical parallel algorithms exist [1, 3, 20, 37, 41, 48], but it remains unknown whether these algorithms are practical (i.e., can be implemented with reasonable engineering effort), and if so, whether they can yield high performance. *The goal of this paper is to formally study parallel solutions for edit distance. By carefully studying existing theoretical solutions, we develop* ***new output-sensitive parallel solutions with good theoretical guarantees and high performance in practice****. We also conduct in-depth experimental studies on existing and our new algorithms.*

The classic dynamic programming (DP) algorithm solves edit distance by using the states $G[i, j]$ as the edit distance of transforming $A[1..i]$ to $B[1..j]$. $G[i, j]$ can be computed as:

$$G[i,j] = \begin{cases} G[i-1,j-1] & \text{if } A[i] = B[j] \text{ and } i > 0, j > 0 \\ 1 + \min(G[i-1,j], G[i-1,j-1], G[i,j-1]) & \text{otherwise} \end{cases}$$

$$G[i,j] = \max(i,j) \qquad \text{if } i = 0 \text{ or } j = 0$$

A simple parallelization of this computation is to compute all states with the same $i + j$ value in parallel, and process all $i + j$ values in an incremental order [7, 55, 58]. However, this approach has low parallelism as it requires $n + m$ rounds to finish. Later work [1, 3, 41]

**Table 1 Algorithms in this paper.** $k$ is the edit distance. $b$ is the block size. $^*$: Monte Carlo algorithms due to the use of hashing. "Space$^*$" means auxiliary space used in addition to the input. Here we assume constant alphabet size for BFS-SA.

| Algorithm | Work | Span | Space$^*$ | Algorithm | Work | Span | Space$^*$ |
|---|---|---|---|---|---|---|---|
| **BFS-SA** | $O(n+k^2)$ | $\tilde{O}(k)$ | $O(n)$ | **BFS-Hash**$^*$ | $O(n+k^2\log n)$ | $\tilde{O}(k)$ | $O(n)$ |
| **DaC-SD** | $O(nk\log k)$ | $\tilde{O}(1)$ | $O(nk)$ | **BFS-B-Hash**$^*$ | $O(n+k^2b\log n)$ | $\tilde{O}(kb)$ | $O(n/b+k)$ |

improved parallelism using a ***divide-and-conquer (DaC)*** approach and achieved $\tilde{O}(n^2)$ work and polylog$(n)$ span. These algorithms use the monotonicity of the DP recurrence, and are complicated. There are two critical issues in the DaC approaches. First, to the best of our knowledge, there exist no implementations given the sophistication of these algorithms. Second, they are not output-sensitive ($\tilde{O}(nm)$ work), which is inefficient when $k \ll n$.

Alternatively, many existing solutions, both sequentially [19, 20, 21, 22, 26, 35, 36, 46, 47] and in parallel [20, 37] use output-sensitive algorithms, and achieve $\tilde{O}(nk)$ or $\tilde{O}(n+k^2)$ work and $\tilde{O}(k)$ span. These algorithms view DP table as a grid-like DAG, where each state (cell) $(x, y)$ has three incoming edges from $(x-1, y)$, $(x, y-1)$, and $(x-1, y-1)$ (if they exist). The edge weight is 0 from $(x-1, y-1)$ to $(x, y)$, when $A[x] = B[y]$, and 1 otherwise. Then edit distance is equivalent to the shortest path from $(0, 0)$ to $(n, m)$. An example is given in Fig. 1. Since the edge weights can only be 0 or 1, we can use ***breadth-first search (BFS)*** from the cell $(0, 0)$ until $(n, m)$ is reached. Ukkonen [56] further showed that using ***longest common prefix (LCP)*** queries based on suffix trees or suffix arrays, the work can be improved to $O(n + k^2)$. Landau and Vishkin [37] parallelized this algorithm (see Sec. 3). While the sequential output-sensitive algorithms have been widely used in practice [21, 26, 36, 46, 47], we are unaware of any existing implementations for the parallel version.

We systematically study parallel output-sensitive edit distance, using both the BFS-based and the DaC-based approaches. Our first effort is to implement the BFS-based Landau-Vishkin algorithm with our carefully-engineered parallel suffix array (SA) implementation, referred to as BFS-SA. Although suffix array is theoretically efficient with $O(n)$ construction work, the hidden constant is large. Thus, we use hashing-based solutions to replace SA for LCP queries to improve the performance in practice. We first present a simple approach BFS-Hash in Sec. 3.2 that stores a hash value for all prefixes of the input. This approach has $O(n)$ construction work, $O(\log n)$ per LCP query, and $O(n)$ auxiliary space. While both BFS-SA and BFS-Hash take $O(n)$ extra space, such space overhead can be significant in practice – for example, BFS-Hash requires $n$ 64-bit hash values, which is 4× the input size considering characters as inputs, and 32× with even smaller alphabet such as molecule bases (alphabet as $\{A, C, G, T\}$). To address the space issue, we proposed BFS-B-Hash using *blocking*. Our solution takes a user-defined parameter $b$ as the block size, which trades off between space usage and query time. BFS-B-Hash limits extra space in $O(n/b)$ by using $O(b \log n)$ LCP query time. Surprisingly, despite a larger LCP cost, our hash-based solutions are consistently faster than BFS-SA in all real-world test cases, due to cheaper construction. All of our BFS-based solutions are simple to program.

We also study the DaC-based approach and propose a parallel output-sensitive solution. We propose a non-trivial adaption for the AALM algorithm [1] to make it output-sensitive. Our algorithm is inspired by the BFS-based approaches, and improves the work from $\tilde{O}(nm)$ to $\tilde{O}(nk)$, with polylogarithmic span. The technical challenge is that the states in the computation are no longer a rectangle, but an irregular shape (see Fig. 1 and 3). We then present a highly non-trivial implementation of this algorithm. Among many key challenges,

we highlight our solution to avoid dynamically allocating arrays in the recursive execution. While memory allocation is mostly ignored theoretically, in practice it can easily be the performance bottleneck in the parallel setting. We refer to this implementation as DaC-SD, with details given in Sec. 4 and 5.2 and the full version of this paper [16].

The bounds of our algorithms (BFS-SA, BFS-Hash, BFS-B-Hash, and DaC-SD) are presented in Tab. 1. We implemented them and show an experimental study in Sec. 6. We tested both synthetic and real-world datasets, including DNA, English text from Wikipedia, and code repositories from GitHub, with string lengths in $10^5$–$10^9$ and varying edit distances, many of them with real edits (e.g., edit history from Wikipedia and commit history on GitHub). In most tests, our new BFS-B-Hash or BFS-Hash performs the best, and their relative performance depends on the value of $k$ and the input patterns. Our BFS-based algorithms are faster than the existing parallel output-insensitive implementation in ParlayLib [7], even with a reasonably large $k \approx 10^5$. We believe that our paper is the first systematic study in theory and practice of parallel edit distance, and we give the first publicly available parallel edit distance implementation that can process *billion-scale strings* with small edit distance and our code at [17]. Due to page limit, some details are provided in the full version of this paper [16]. We summarize our contributions as follows:

1. Two new BFS-based edit distance solutions BFS-Hash and BFS-B-Hash using hash-based LCP queries. Compared to the existing SA-based solution in Landau-Vishkin, our hash-based solutions are simpler and more practical. BFS-B-Hash also allows for tradeoffs between time and auxiliary space.
2. A new DaC-based edit distance solution DaC-SD with $O(nk \log k)$ work and polylogarithmic span.
3. New implementations for four output-sensitive edit distance algorithms: BFS-SA, BFS-Hash, BFS-B-Hash and DaC-SD. Our code is publicly available [17].
4. Experimental study of the existing and our new algorithms on different input patterns.

## 2 Preliminaries

We use $O(f(n))$ *with high probability* (*whp*) (in $n$) to mean $O(cf(n))$ with probability at least $1 - n^{-c}$ for $c \geq 1$. We use $\tilde{O}(f(n))$ to denote $O(f(n) \cdot \text{polylog}(n))$. For a string $A$, we use $A[i]$ as the $i$-th character in $A$. We use *string* and *sequence* interchangeably. We use $A[i..j]$ to denote the $i$-th to the $j$-th characters in $A$, and $A[i..j)$ the $i$-th to the $(j-1)$-th characters in $A$. Throughout the paper, we use "auxiliary space" to mean space used in addition to the input.

**String Edit Distance.** Given two strings $A[1..n]$ and $B[1..m]$, Levenshtein's Edit Distance [38] between $A$ and $B$ is the minimum number of operations needed to convert $A$ to $B$ by using insertions, deletions, and substitutions. We also call the operations ***edits***. In this paper, we use ***edit distance*** to refer to Levenshtein's Edit Distance. The classic dynamic programming (DP) algorithm for edit distance uses DP recurrence shown in Sec. 1 with $O(mn)$ work and space.

**Hash Functions.** For the simplicity of algorithm descriptions, we assume a perfect hash function for string comparisons, i.e., a function $h : S \to [1, O(|S|)]$ such that $h(x) = h(y) \iff x = y$. For any alphabet $\Sigma$ with size $|\alpha|$, we use a hash function $h(A[l..r]) = \sum_{i=l}^{r} A[i] \times p^{r-i}$ for some prime numbers $p > |\alpha|$, which returns a unique hash value of the substring $A[l..r]$. The hash values of two consecutive substrings $S_1$ and $S_2$ can be concatenated as $h([S_1, S_2]) =$

$h(S_1)\cdot p^{|S_2|}+h(S_2)$, and the inverse can also be computed as $h(S_2)=h([S_1,S_2])-p^{|S_2|}\cdot h(S_1)$. For simplicity, we denote concatenation and its inverse operation as $\oplus$ and $\ominus$, respectively, as $h([S_1,S_2])=h(S_1)\oplus h(S_2)$ and $h(S_2)=h([S_1,S_2])\ominus h(S_1)$. We assume perfect hashing for theoretical analysis. In practice, we use $p$ as a large prime and modular arithmetic to keep the word-size hash values. In our experiment, we compare different approaches and validate that our implementations are correct in all test cases. However, collisions are possible for other datasets, since different strings may be mapped to the same hash value. If such cases arise, one can either use multiple hash functions for a better success rate in practice, or use the idea of Hirschberg's algorithm [27] to generate the edit sequence and run a correctness check (and restart with another hash function if failed).

**Longest Common Prefix (LCP).** For two sequences $A[1..n]$ and $B[1..m]$, the Longest Common Prefix (LCP) query at position $x$ in $A[1..n]$ and position $y$ in $B[1..m]$ is the longest substring starting from $A[x]$ that match a prefix starting from $B[y]$. With clear context, we also use the term "LCP" to refer to the length of the LCP, i.e., $LCP(A,B,x,y)$ is the length of the longest common prefix substring starting from $A[x]$ and $B[y]$ for $A$ and $B$.

**Computational Model.** We use the ***work-span model*** in the classic multithreaded model with ***binary-forking*** [2, 8, 9]. We assume a set of threads that share the memory. Each thread acts like a sequential RAM plus a `fork` instruction that forks two child threads running in parallel. When both child threads finish, the parent thread continues. A parallel-for is simulated by `fork` for a logarithmic number of steps. A computation can be viewed as a DAG (directed acyclic graph). The ***work $W$*** of a parallel algorithm is the total number of operations, and the ***span (depth) $S$*** is the longest path in the DAG. The randomized work-stealing scheduler can execute such a computation in $W/P+O(S)$ time *whp* in $W$ on $P$ processors [2, 9, 25].

**Suffix Array.** The suffix array (SA) [42] is a lexicographically sorted array of the suffixes of a string, usually used together with the longest common prefix (LCP) array, which stores the length of LCP between every adjacent pair of suffixes. The SA and LCP array can be built in parallel in $O(n)$ work and $O(\log^2 n)$ span *whp* [32, 53].

In edit distance, we need the LCP query between $A[x..n]$ and $B[y..m]$ for any $x$ and $y$. This can be computed by building the SA and LCP arrays for a new string $C[1..n+m]$ that concatenates $A[1..n]$ and $B[1..m]$. The LCP between any pair of suffixes in $C$ can be computed by a range minimum query (RMQ) on the LCP array, which can be built in $O(n+m)$ work and $O(\log(n+m))$ span [8]. Combining all pieces gives the following theorem:

▶ **Lemma 1.** *Given two strings $A[1..n]$ and $B[1..m]$, using a suffix array, the longest common prefix (LCP) between any two substrings $A[x..n]$ and $B[y..m]$ can be reported in $O(1)$ work and span, with $O(n+m)$ preprocessing work and $O(\log^2(n+m))$ span* whp.

# 3 BFS-based Algorithms

## 3.1 Overview of Existing Sequential and Parallel BFS-based Algorithms

Many existing output-sensitive algorithms [19, 20, 21, 22, 26, 35, 36, 37, 46, 47] are based on breadth-first search (BFS). These algorithms view the DP matrix for edit distance as a DAG, as shown in Fig. 1. In this section, we use $x$ and $y$ to denote the row and column

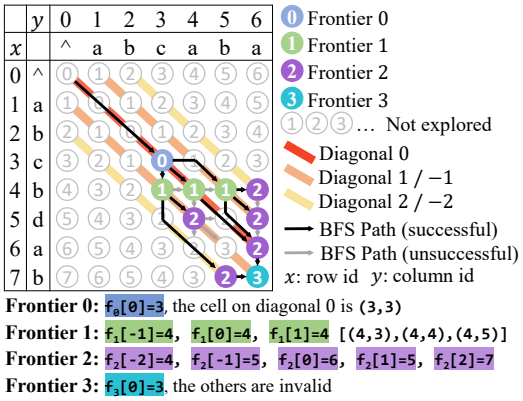



**Figure 1 BFS-based edit distance on $A[1..n]$ and $B[1..m]$.** A more detailed description is in the full version [16]. $f_t[i]$ is the row-id of the last cell on diagonal $i$ with edit distance $t$ (frontier $t$), representing cell $(f_t[i], f_t[i]-i)$.

**Algorithm 1** BFS-based parallel edit distance [37].

```
1  f_0[0] ← LCP(A[1..n], B[1..m])   // Starting point
2  t ← 0
3  while f_t[n − m] ≠ n do
4  |  t ← t + 1
   |  // Find new frontier for diagonal i
5  |  parallel-for-each −t ≤ i ≤ t do
6  |  |  f_t[i] ← f_{t−1}[i]   // Start from the last cell
7  |  |  foreach ⟨dx, dy⟩ ∈ {⟨0,1⟩, ⟨1,0⟩, ⟨1,1⟩} do
   |  |  |  // The previous cell is from diagonal j
   |  |  |  // j = (x − dx) − (y − dy) = i − dx + dy
8  |  |  |  j ← i − dx + dy
9  |  |  |  if |j| ≤ t − 1 then
10 |  |  |  |  The row id x ← f_{t−1}[j] + dx
11 |  |  |  |  The column id y ← x − i
   |  |  |  |  // Skip the common prefix
12 |  |  |  |  x ← x+LCP(A[x+1..n], B[y+1..m])
   |  |  |  |  // Keep the largest row id
13 |  |  |  |  f_t[i] ← max(f_t[i], x)
14 return t
```

ids of the cells in the DP matrix, respectively. Each state (cell) $(x, y)$ has three incoming edges from $(x-1, y)$, $(x, y-1)$, and $(x-1, y-1)$ (if they exist). The edge weight is 0 from $(x-1, y-1)$ to $(x, y)$ when $A[x] = B[y]$, and 1 otherwise. Then edit distance is equivalent to the shortest distance from $(0, 0)$ to $(n, m)$.

Since the edge weights are 0 or 1, we can use a special breadth-first search (BFS) to compute the shortest distance. In round $t$, we process states with edit distance $t$. The algorithm terminates when we reach cell $(n, m)$. First observed by Ukkonen [56], in the BFS-based approach, not all states need to be visited. For example, all states with $|x-y| > k$ will not be reached before we reach $(n, m)$ with edit distance $k$, since they require more than $k$ edits. Thus, this BFS will touch at most $O(kn)$ cells, leading to $O(kn)$ work.

Another key observation is that starting from any cell $(x, y)$, if there are diagonal edges with weight 0, we should always follow the edges until a unit-weight edge is encountered. Namely, we should always find the longest common prefix (LCP) from $A[x+1]$ and $B[y+1]$, and skip to the cell at $(x+p, y+p)$ with no edit, where $p$ is the LCP length. This idea is used in Landau and Vishkin [37] on parallel approximate string matching, and we adapt this idea to edit distance here. Using the modified parallel BFS algorithm by Landau-Vishkin [37] (shown in Alg. 1), only $O(k^2)$ states need to be processed – on each diagonal and for each edit distance $t$, only the last cell with $t$ edits needs to be processed (see Fig. 1). Hence, the BFS runs for $k$ rounds on $2k+1$ diagonals, which gives the $O(k^2)$ bound above. In the BFS algorithm, we can label each diagonal by the value of $x-y$. In round $t$, the BFS visits a *frontier* of cells $f_t[\cdot]$, where $f_t[i]$ is the cell with edit distance $t$ on diagonal $i$, for $-t \le i \le t$. We present the algorithm in Alg. 1 and an illustration in Fig. 1. Note that in the implementation, we only need to maintain two frontiers (the previous and the current one), which requires $O(k)$ space. We provide more details about this algorithm in the full version [16]. If the LCP query is supported by suffix arrays, we can achieve $O(n+k^2)$ work and $O(\log n + k \log k)$ span for the edit distance algorithm.

**Algorithm Based on Suffix Array (BFS-SA).** Using the SA algorithm in [32] and the LCP algorithm in [53] for Landau-Vishkin gives the claimed bounds in Tab. 1. We present details about our SA implementation in Sec. 5.1.

## 3.2 Algorithm Based on String Hashing (BFS-Hash)

Although BFS-SA is theoretically efficient with $O(n)$ preprocessing work to construct the SA, the hidden constant is large. For better performance, we consider string hashing as an alternative for SA. Similar attempts (e.g., locality-sensitive hashing) have also been used in approximate pattern matching problems [43, 44]. In our pursuit of exact output-sensitive edit distance computation, we draw inspiration from established string hashing algorithms, such as the Rabin-Karp algorithm (also known as rolling hashing) [33]. We will first present a simple hash-based solution BFS-Hash with $O(n)$ preprocessing cost and $O(n)$ auxiliary space. Then later in Sec. 3.3, we will present BFS-B-Hash, which saves auxiliary space by trading off more work in LCP queries.

As mentioned in Sec. 2, the hash function $h(\cdot)$ maps any substring $A[l..r]$ to a unique hash value, which provides a fingerprint for this substring in the LCP query. The high-level idea is to binary search the query length, using the hash value as validation. We precompute the hash values for all prefixes, i.e., $T_A[x] = h(A[1..x])$ for the prefix substring $A[1..x]$ (similar for $B$). They can be computed in parallel by using any scan (prefix-sum) operation [6] with $O(n)$ work and $O(\log n)$ span. We can compute $h(A[l..r])$ by $T_A[r] \ominus T_A[l-1]$.

With the preprocessed hash values, we dual binary search the LCP of $A[x..n]$ and $B[y..m]$. We compare the hash values starting from $A[x]$ and $B[y]$ with chunk sizes of $1, 2, 4, 8, \ldots$, until we find value $l$, such that $A[x..x+2^l) = B[y..y+2^l)$, but $A[x..x+2^{l+1}) \neq B[y..y+2^{l+1})$. By doing this with $O(\log n)$ work, we know that the LCP of $A[x..n]$ and $B[y..m]$ must have a length in the range $[2^l, 2^{l+1})$. We then perform a regular binary search in this range, which costs another $O(\log n)$ work. This indicates $O(\log n)$ work in total per LCP query. Combining the preprocessing and query costs, we present the cost bounds of BFS-Hash:

▶ **Theorem 2.** *BFS-Hash computes the edit distance between two sequences of length $n$ and $m \leq n$ in $O(n + k^2 \log n)$ work, $\tilde{O}(k)$ span, and $O(n)$ auxiliary space, where $k$ is the output size (fewest possible edits).*

BFS-Hash is simple and easy to implement. Our experimental results indicate that its simplicity also allows for a reasonably good performance in practice for most real-world input instances. However, this algorithm uses $n$ 64-bit integers as hash values, and such space overhead may be a concern in practice. This is more pronounced when the input is large and/or the alphabet is small (particularly when each input element can be represented with smaller than byte size), as the auxiliary space can be much larger than the input size. This concern also holds for BFS-SA as several $O(n)$-size arrays are needed during SA construction. Note that for shared-memory parallel algorithms, space consumption is also a *key constraint* – if an algorithm is slow, we can wait for longer; but if data (and auxiliary data) do not fit into the memory, then this algorithm is not applicable to large input at all. In this case, the problem size that is solvable by the algorithm is limited by the space overhead, which makes the improvement from parallelism much narrower. Below we will discuss how to make our edit distance algorithms more space efficient.

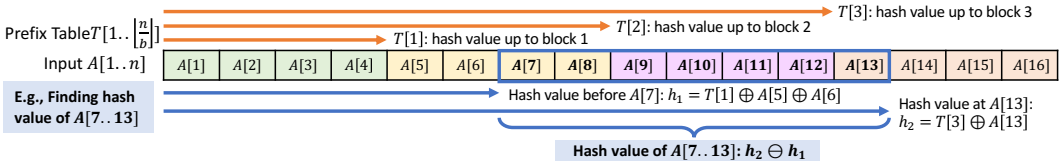



**Figure 2 The illustrations of prefix table values and one specific query**, with key concepts shown when computing the hash value of a range using a prefix table.

**Algorithm 2** The prefix table for finding the longest common prefix of $A[1..n]$ and $B[1..m]$.

```
1  // Table construction
2  Function Construct(A, B)
3      T_A[·] ← Build(A)
4      T_B[·] ← Build(B)
   // The prefix table building process
5  Function Build(A)
6      w ← ⌊|A|/b⌋
7      T[0] ← 0
8      parallel-for-each j ← 1 to w do
9          T[j] ← h(A[(j − 1)b + 1 .. jb])
10     Scan(T)
11     return T[·]
   // Get hash value for prefix sub-
   // sequence A[1..x]
12 Function GetHash(A, T_A, x)
13     if x = 0 then return 0
14     r ← ⌊(x − 1)/b + 1⌋
15     h̄ ← T_A[r]
16     for i ← r · b + 1 to x do
17         h̄ ← h̄ ⊕ h(A[i])
18     return h̄
   // Compare the subsequences A[x..x + l] and B[y..y + l]
19 Function Compare(A, B, x, y, l)
20     h_A ← GetHash(A, T_A, x + l) ⊖ GetHash(A, T_A, x − 1)
21     h_B ← GetHash(B, T_B, y + l) ⊖ GetHash(B, T_B, y − 1)
22     return h_A = h_B
   // Longest Common Prefix from A[x] and B[y]
23 Function LCP(A, B, x, y)
24     l_1 ← 0
       // Find l_1, s.t. the LCP is between 2^{l_1} to 2^{l_1+1}
       blocks
25     while x + 2^{l_1} < n and y + 2^{l_1} < m do
26         if Compare(A, B, x, y, l_1) = false then break
27         l_1 ← l_1 + 1
       // Trivial binary search process on the range
       [2^{l_1}, 2^{l_1+1})
28     s ← 2^{l_1}, t ← 2^{l_1+1}
29     while s < t do
30         if Compare(A, B, x, y, ⌊(s + t)/2⌋) = false then
31             t ← ⌊(s + t)/2⌋
32         else s ← ⌊(s + t)/2⌋ + 1
33     return s
```

## 3.3 Algorithm Based on Blocked-Hashing (BFS-B-Hash)

In this section, we introduce our BFS-B-Hash algorithm that provides a more space-efficient solution by trading off worst-case time (work and span). Interestingly, we observed that on many data sets, BFS-B-Hash can even outperform BFS-Hash and other opponents due to faster construction time, and we will analyze that in Sec. 6.

To achieve better space usage, we divide the strings into blocks of size $b$. As such, we only need to store the hash values for prefixes of the entire blocks $h(A[1..b]), h(A[1..2b]), \cdots, h(A[1..\lfloor (n/b) \rfloor \cdot b])$. Our idea of blocking is inspired by many string algorithms (e.g., [4]). Using this approach, we only need auxiliary space to store $O(n/b)$ hash values, and thus we can control the space usage using the parameter $b$. To compute these hash values, we will first compute the hash value for each block, and run a parallel scan (prefix sum on $\oplus$) on the hash values for all the blocks. Similar to the above, we refer to these arrays as $T_A[i] = h(A[1..ib])$ (and $T_B[i]$ accordingly), and call them ***prefix tables***.

We now discuss how to run LCP with only partial hash values available. The *LCP* function in Alg. 2 presents the process to find the LCP of $A[x..n]$ and $B[y..m]$ using the prefix tables. We present an illustration in Fig. 2. We will use the same dual binary search approach to find the LCP of two strings. Since we do not store the hash values for all prefixes, we use a function $GetHash(A, T_A, x)$ to compute $h(A[1..x])$. We can locate the

closest precomputed hash value and use $r$ as the previous block id before $x$. Then the hash value up to block $r$ is simply $\bar{h} = T_A[r]$. We then concatenate the rest characters to the hash value (i.e., return $\bar{h} \oplus h(A[rb+1]) \oplus \cdots \oplus h(A[x])$). In this way, we can compute the hash value of any prefixes for both $A$ and $B$, and plug this scheme into the dual binary search in BFS-Hash. In each step of dual binary search, the concatenation of hash value can have at most $b$ steps, and thus leads to a factor of $b$ overhead in query time than BFS-Hash.

▶ **Theorem 3.** *BFS-B-Hash computes the edit distance between two sequences of length $n$ and $m \le n$ in $O(n + k^2 \cdot b \log n)$ work and $\tilde{O}(kb)$ span, using $O(n/b + k)$ auxiliary space, where $k$ is the output size (fewest possible edits).*

The term $k$ in space usage is from the BFS (each frontier is at most size $O(k)$). $O(b \log n)$ is the work for each LCP query. Note that this is an upper bound – if the LCP length $L$ is small, the cost can be significantly smaller (a tighter bound is $O(\min(L, b \log L))$). Sec. 6 will show that for normal input strings where the LCP lengths are small in most queries, the performance of BFS-B-Hash is indeed the fastest, although for certain input instances when the worst case is reached, the performance is not as good.

## 4 The Divide-and-Conquer Algorithms

Our parallel output-sensitive algorithm DaC-SD is inspired by the AALM algorithm [1], and also uses it as a subroutine. We first overview the AALM algorithm, and introduce our algorithm in details. We assume $m = n$ is a power of 2 in this section for simple descriptions, but both our algorithm and AALM work for any $n$ and $m$.

**The AALM Algorithm.** As described above, the edit distance problem can be considered as a shortest distance (SD) problem from the top-left cell $(0, 0)$ to the bottom-right cell $(n, n)$ in the DP matrix $G$. Instead of directly computing the SD from $(0, 0)$ to $(n, n)$, AALM computes pairwise SD between any cell on the left/top boundaries and the bottom/right boundaries (i.e., those on $L \cup U$ to $W \cup R$ in Fig. 3(a)). We relabel all cells in $L \cup U$ as a sequence $v = \{v_0, v_1, \ldots v_{2n}\}$ (resp., $W \cup R$ as $u = \{u_0, u_1 \cdots, u_{2n}\}$), as shown in Fig. 3. Therefore, for the DP matrix $G$, the pairwise SD between $v$ and $u$ forms a $(2n+1) \times (2n+1)$ matrix. We call it the ***SD matrix*** of $G$, and denote it as $D_G$. AALM uses a divide-and-conquer approach. It first partitions $G$ into four equal submatrices $G_1$, $G_2$, $G_3$, and $G_4$ (See Fig. 3(b)), and recursively computes the SD matrices for all $G_i$. We use $D_i$ to denote the SD matrix for $G_i$. In the "conquer" step, the AALM algorithm uses a *Combine* subroutine to combine two SD matrices into one if they share a common boundary (our algorithm also uses this subroutine). For example, consider combining $G_1$ and $G_2$. We still use $v_i$ and $u_j$ to denote the cells on the left/top and bottom/right boundaries of $\begin{pmatrix} G_1 \\ G_2 \end{pmatrix}$ (see Fig. 3(c)), and denote the cells on the common boundary of $G_1$ and $G_2$ as $w_1, \cdots, w_{n/2}$, ordered from left to right. For any pair $v_i$ and $u_j$, if they are in the same submatrix, we can directly get the SD from the corresponding SD matrix. Otherwise, WLOG assume $v_i \in G_1$ and $u_j \in G_2$, then we compute the SD between them by finding $\min_l D_1[i, l] + D_2[l, j]$, i.e., for all $w_l$ on the common boundary, we attempt to use the SD between $v_i$ to $w_l$, and $w_l$ to $u_j$, and find the minimum one. Similarly, we can combine $D_3$ with $D_4$, and $D_{1\cup2}$ with $D_{3\cup4}$, and eventually get $D_G$. We note that the *Combine* algorithm, even theoretically, is highly involved. At a high level, it uses the Monge property of the shortest distance (the monotonicity of the DP recurrence), and we refer the readers to [1] for a detailed algorithm description and theoretical analysis. In Sec. 5.2,

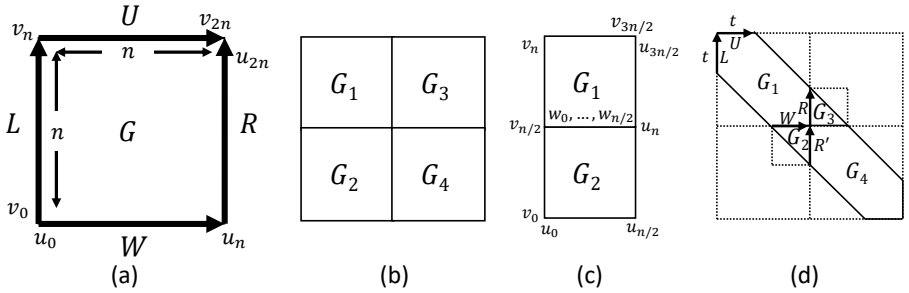



**Figure 3 The illustrations of the key concepts and notation in the AALM algorithm described in Sec. 4.**

we highlight a few challenges and our solutions for implementing this highly complicated algorithm. Theoretically, combining two $n \times n$ SD matrices can be performed in $O(n^2)$ work and $O(\log^2 n)$ span, which gives $O(n^2 \log n)$ work and $O(\log^3 n)$ span for AALM.

**Our algorithm.** The AALM algorithm has $\tilde{O}(n^2)$ work ($\tilde{O}(nm)$ if $n \neq m$) and polylogarithmic span, which is inefficient in the output-sensitive setting. As mentioned in Sec. 3.1, only a narrow width-$O(k)$ diagonal area in $G$ is useful (Fig. 3(d)). We thus propose an output-sensitive DAC-SD algorithm adapted from the AALM algorithm. We follow the same steps in AALM, but restrict the paths to the diagonal area, although the exact size is unknown ahead of time. We first present the algorithm to compute the shortest distance on the diagonal region with width $2t+1$ as function *Check*($t$) in Alg. 3, which restricts the search in diagonals $-t$ to $t$. First, we divide such a region into four sub-regions (see Fig. 3(d)). Two of them ($G_1$ and $G_4$) are of the same shape, and the other two of them ($G_2$ and $G_3$) are triangles. For $G_2$ and $G_3$, we use the AALM algorithm to compute their SD matrices by aligning them to squares. For $G_1$ and $G_4$, we process them recursively, until the base case where the edge length of the matrix is smaller than $t$ and they degenerate to squares, in which case we apply the AALM algorithm. Note that even though the width-$(2t+1)$ diagonal stripe is not a square ($G_1$ and $G_4$ are also of the same shape), the useful boundaries are still the left/top and bottom/right boundaries ($L \cup U$ and $W \cup R$ in Fig. 3(d)). Therefore, we can use the same *Combine* algorithm as in AALM to combine the SD matrices. For example, in Fig. 3(d), when combining $G_1$ with $G_2$, we obtain the pairwise distance between $L \cup U$ and $R \cup R'$ using the common boundary $W$. We can similarly combine all $G_1, G_2, G_3$, and $G_4$ to get the SD matrix for $G$.

However, the output value $k$ is unknown before we run the algorithm. To overcome this issue, we use a strategy based on prefix doubling to "binary search" the value of $k$ without asymptotically increasing the work of the algorithm. We start with $t = 1$, and run the *Check*($t$) in Alg. 3 (i.e., restricting the search in a width-$(2t+1)$ diagonal). Assume that the *Check* function returns $\sigma$ edits. If $\sigma \leq t$, we know that $\sigma$ is the SD from $(0,0)$ to $(n,n)$, since allowing the path to go out of the diagonal area will result in an answer greater than $t$. Otherwise, we know $\sigma > t$, and $\sigma$ is not necessarily the shortest distance from the $(0,0)$ to $(n,n)$, since not restricting the path in the $t$-diagonal area may allow for a shorter path. If so, we double $t$ and retry. Although we need $O(\log k)$ searches before finding the final answer $k$, we will show that the total search cost is asymptotically bounded by the last search. In the last search, we have $t < 2k$.

**Algorithm 3** Divide-and-Conquer edit distance algorithm on $A[1..n]$ and $B[1..n]$.

```
1  Notes: We assume both A and B has size n = 2^c for simple description. Our
       algorithm also works for strings with different lengths with minor changes.
2  Function DAC-SD
3      t ← 1
4      while true do
5          D ← Check(t)
6          if D[t][t] ≤ t then break
7          t ← min(2t, n)
8      return D[t][t]
   // Find the SD in the DP matrix from (0,0) to (n,n) by restricting in the diagonal stripe with width t
9  Function Check(t)
10     D ←GetDistance(0, 0, n, t)
Function GetDistance(i, j, n, t)
    if n/2 < t then
        Computed D by the AALM algorithm
        return D
    // Compute the SD matrices for G1, G2, G3, G4 (as shown in Fig. 3 (d)).
11  D1 ← GetDistance(i, j, n/2, t)
12  Compute D2 and D3 by the AALM algorithm
13  D4 ← GetDistance(i + n/2, j + n/2, n − n/2, t)
    // use the same Combine function as AALM
14  D1∪2 ← Combine(D1, D2)
15  D3∪4 ← Combine(D3, D4)
16  D ← (D1∪2, D3∪4)
17  return D
```

We first analyze the cost for $Check(t)$. It contains two recursive calls, two calls to AALM, and three calls to the *Combine* function. Therefore, the work for $Check(t)$ is $W(n) = 2W(n/2) + O(t^2 \log t)$, with base cases $W(t) = t^2 \log t$, which solves to $W(n) = O(nt \log t)$. For span, note that there are $\log(n/t)$ levels of recursion before reaching the base cases. In each level, the *Combine* function combines $t \times t$ SD matrices with $O(\log^2 t)$ span. In the leaf level, the base case uses AALM with $O(\log^3 t)$ span. Therefore, the total span of a *Check* is:

$$O(\log n/t \cdot \log^2 t + (\log n/t + \log^3 t)) = O(\log^2 t \cdot (\log n/t + \log t)) = O(\log^2 t \log n) \quad (1)$$

We will apply $Check(\cdot)$ for $O(\log k)$ times, with $t = 1, 2, 4, \ldots$ up to at most $2k$. Therefore, the total work is dominated by the last *Check*, which is $O(nk \log k)$. The span is $O(\log n \log^3 k)$.

▶ **Theorem 4.** *The DAC-SD algorithm computes the edit distance between two sequences of length $n$ and $m \le n$ in $O(nk \log k)$ work and $O(\log n \log^3 k)$ span, where $k$ is the output size (fewest possible edits).*

Compared to the BFS-based algorithms with $\tilde{O}(k)$ span, our DAC-SD is also output-sensitive and achieves polylogarithmic span. However, the work is $\tilde{O}(kn)$ instead of $\tilde{O}(n + k^2)$, which will lead to more running time in practice for a moderate size of $k$.

## 5 Implementation Details

We provide all implementations for the four algorithms as well as testing benchmarks at [17]. In this section, we highlight some interesting and challenging parts of our implementations.

### 5.1 Implementation Details of BFS-based Algorithms

For the suffix array construction in BFS-SA, we implemented a parallel version of the DC3 algorithm [32]. We also compared our implementation with the SA implementation in ParlayLib [7], which is a highly optimized version of the prefix doubling algorithm with $O(n \log n)$ work and $O(\log^2 n)$ span. On average, our implementation is about $2\times$ faster than that in ParlayLib when applied to edit distance. We present some results for their comparisons in the full version [16]. For LCP array construction and preprocessing RMQ queries, we use the implementation in ParlayLib [7], which requires $O(n \log n)$ work and $O(\log^2 n)$ span. With them, the query has $O(1)$ cost.

In our experiments on both synthetic and real-world data, we observed that the LCP length is either very large when we find two long matched chunks, or in most of the cases, very short when they are not corresponding to each other. This is easy to understand – for genomes, text or code with certain edit history, it is unlikely that two random starting positions share a large common prefix. Based on this, we add a simple optimization for all LCP implementations such that we first compare the leading eight characters, and only when they all match, we use the regular LCP query. This simple optimization greatly improved the performance of BFS-SA, and also slightly improved the hash-based solutions.

## 5.2 Implementation Details of the DaC-SD Algorithm

Although our DaC-SD algorithm given in Alg. 3 is not complicated, we note that implementing it is highly non-trivial in two aspects. First, in Sec. 4, we assume both strings $A$ and $B$ have the same length $n$, which is a power of two. However, handling two strings with different lengths makes the matrix partition more complicated in practice. Another key challenge is that the combining step in the AALM algorithm is recursive and needs to allocate memory with varying sizes in the recursive execution. While memory allocation is mostly ignored theoretically, frequent allocation in practice can easily be the performance bottleneck in the parallel setting. We discuss our engineering efforts as follows.

**Irregularity.** The general case, when $n$ and $m$ are not powers of two and not the same, is more complicated than the case in Alg. 3. In this case, all four subproblems $G_1$, $G_2$, $G_3$, and $G_4$ will have different sizes. While theoretically, we can always round up, for better performance in practice, we need to introduce additional parameters to restrict the search within the belt region as shown in Fig. 4. Therefore, we use two parameters $t_1$ and $t_2$, to denote the lengths of the diagonal area on each side. We show an illustration in Fig. 4(a) along with how to compute the subproblem sizes. In extreme cases, $t_1$ or $t_2$ can degenerate to 0, which results in three subproblems (Fig. 4(b)). In such cases, we will first merge $G_2$ and $G_4$, then merge $G_1$ and $G_{2\cup4}$.

**The Combining Step.** As mentioned in Sec. 4, achieving an efficient combining step is highly non-trivial. The straightforward solution to combine two matrices is to use the Floyd-Warshall algorithm [18], but it incurs $O(n^3)$ work and will be a bottleneck. The AALM algorithm improves this step to $O(n^2)$ by taking advantage of the Monge property of the two matrices. For page limit, we introduce the details of the combining algorithm in the full version [16]. However, the original AALM algorithm is based on divide-and-conquer and requires memory allocation for every recursive function call. This is impractical as frequent parallel memory allocation is extremely inefficient. To overcome this challenge, we redesign the recursive solution to an iterative solution, such that we can preallocate the memory space before the combining step. No dynamic memory allocation is involved during the computation. We provide the details of this approach in the full version [16].

# 6 Experiments

**Setup.** We implemented all algorithms in C++ using ParlayLib [7] for fork-join parallelism and some parallel primitives (e.g., reduce). Our tests use a 96-core (192 hyperthreads) machine with four Intel Xeon Gold 6252 CPUs, and 1.5 TB of main memory. We utilize `numactl -i all` in tests with more than one thread to spread the memory pages across CPUs in a round-robin fashion. We run each test three times and report the median.

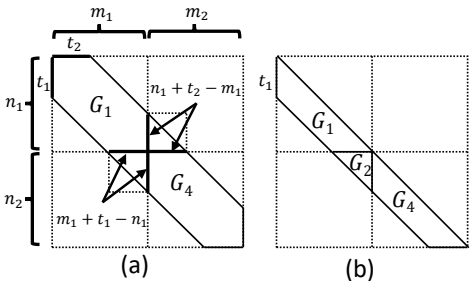



**Figure 4 The illustrations of our output-sensitive DaC-SD algorithm.** (a) Two parameters $t_1$ and $t_2$ are needed to denote the lengths of the diagonal area on each side. (b) The case that $t_2 = 0$ and $G_3$ degenerates.

**Table 2 Real-world datasets in our experiments, including input sizes $|A|$ and $|B|$, number of edits $k$, and alphabet sizes $|\Sigma|$.**

| Data | Alias | $\|A\|$ | $\|B\|$ | $k$ | $\|\Sigma\|$ |
|---|---|---|---|---|---|
| Wikipedia pages [45] | Wiki v1 | 0.56M | 0.56M | 439 | 256 |
| | Wiki v2 | 0.56M | 0.56M | 5578 | 256 |
| | Wiki v3 | 0.56M | 0.55M | 15026 | 256 |
| Linux kernel code [40] | Linux v1 | 6.47M | 6.47M | 236 | 256 |
| | Linux v2 | 6.47M | 6.47M | 1447 | 256 |
| | Linux v3 | 6.47M | 6.46M | 9559 | 256 |
| DNA sequences [5] | DNA 1 | 42.3M | 42.3M | 928 | 4 |
| | DNA 2 | 42.3M | 42.3M | 9162 | 4 |
| | DNA 3 | 42.3M | 42.3M | 91419 | 4 |

**Tested Algorithms and Datasets.** We tested five algorithms in total: four output-sensitive algorithms in this paper (BFS-SA, BFS-Hash, BFS-B-Hash, DaC-SD), and a baseline algorithm from ParlayLib [7], which is a parallel output-insensitive implementation with $O(nm)$ work. The ParlayLib implementation is intended to showcase the simplicity of parallel algorithms, and as a result, it may not be well-ptimized. We are unaware of other parallel implementations that provide output-sensitive cost bounds. We use $b = 32$ for our BFS-B-Hash. As we will show later, the running time is generally stable with $4 \le b \le 64$. We tested the algorithms on both synthetic and real-world datasets. For synthetic datasets, we generate random strings with different string lengths $n = 10^i$ for $6 \le i \le 9$ and $k$ (number of edits) varying from 1 to $10^5$, and set the size of the alphabet as 256. We create strings $A$ and $B$ by generating $n$ random characters, and applying $k$ edits. The $k$ edits are uniformly random for insertion, deletion and substitution. For $k \ll n$, we have $m \approx n$. All values of $k$ shown in the figures and tables are approximate values. Our real-world datasets include Wikipedia [45], Linux kernel [40], and DNA sequences [51]. We compare the edit distance between history pages on Wikipedia and history commits of a Linux kernel file on GitHub. We also compare DNA sequences by adding valid modifications to them to simulate DNA damage or genome editing techniques, as is used in many existing papers [11, 13, 30, 57]. We present the statistics of the real-world datasets in Tab. 2.

**Overall Performance on Synthetic Data.** We present our results on synthetic data in the upper part of Fig. 5. We also present the complete results in the full version [16]. For BFS-based algorithms, we also separate the time for ***building*** the data structures for LCP queries, and the ***query*** time (the BFS process). ParlayLib cannot process instances with $n > 10^6$ due to its $O(nm)$ work bound.

We first *compare our solutions with ParlayLib* [7]. Since ParlayLib is not output-sensitive, its running time remains the same regardless of the value of $k$. Among the tests that ParlayLib can process ($n = 10^6$), our output-sensitive algorithms are much faster than ParlayLib, especially when $k$ is small (up to $10^5\times$). For $n = 10^6$, all our BFS-based algorithms are at least 1.7× faster than ParlayLib even when $k \approx n/10$.

We then *compare our DaC- and BFS-based solutions.* DaC-SD has the benefit of polylogarithmic span, compared to $\tilde{O}(k)$ span for the BFS-based algorithm. Although this seems to suggest that DaC-SD should have better performance when $k$ is large, the result

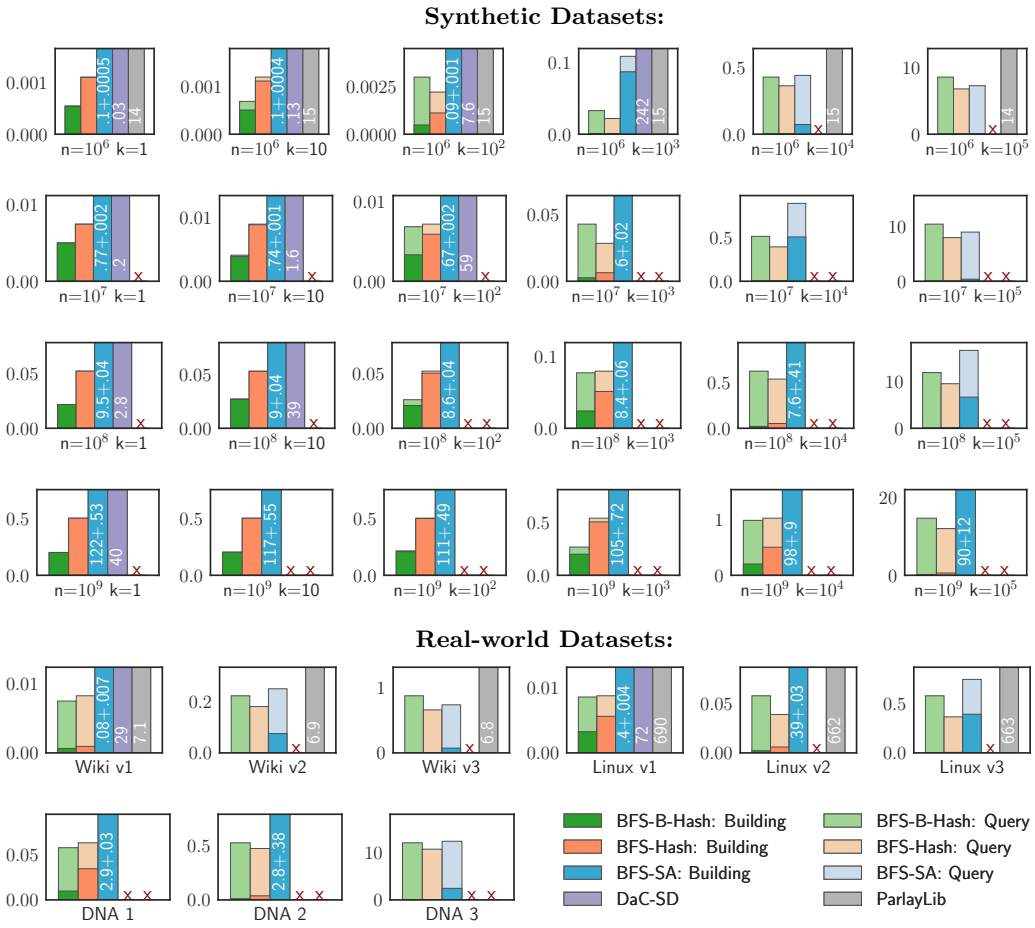
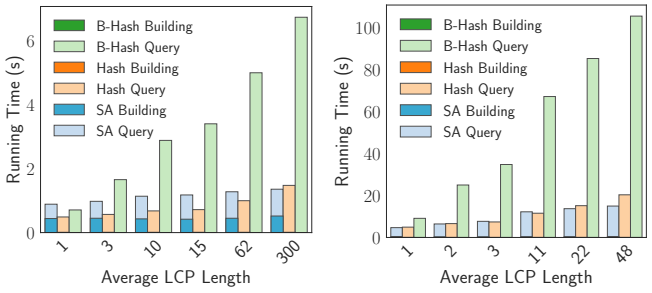



**Figure 5 Running time (in seconds) of synthetic and real-world datasets for all algorithms.** Lower is better. We put an "×" if the algorithm does not finish within 1000 seconds. For BFS-based algorithms, we separate the time into building time (constructing the data structure for LCP queries) and query time (running BFS). All bars out of the range of the y-axis are annotated with numbers. The number is the total running time for DaC-SD and ParlayLib, and is in the format of $a + b$ for BFS-SA, where $a$ is the building time and $b$ is the query time. Full results are presented in the full version [16].

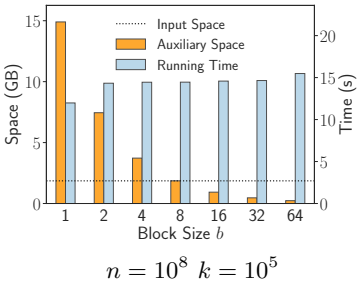
shows the opposite. The reason is that DaC-SD has $\tilde{O}(nk)$ work, compared to $\tilde{O}(n + k^2)$ cost of the BFS-based algorithms. When $k$ becomes larger, the overhead in work is also more significant. On the other hand, when $k$ is small, the $O(nk)$ work becomes linear, which hides the inefficiency in work. Therefore, the gap between DaC-SD and other algorithms is smaller when $k$ is small, but DaC-SD is still slower than BFS-based algorithms in all test cases, especially when $k$ is large. This experiment reaffirms the *importance of work efficiency* on practical performance for parallel algorithms.

Finally, we *compare all our BFS-based solutions.* Our hash-based solutions have significant advantages over the other implementations when $k$ is small, since the pre-processing time for hash-based solutions is much shorter. When $k$ is large, pre-processing time becomes negligible, and BFS-Hash seems to be the ideal choice since its query is also efficient. In particular, for $n \approx m \approx 10^9$, hash-based algorithms use about 1 second for pre-processing while BFS-SA uses about 100 seconds. Although BFS-SA also has $O(n)$ construction time,




**(a)** $n = 10^7$ $k = 10000$. **(b)** $n = 10^7$ $k = 50000$.

**Figure 6 Performance of BFS-based algorithms vs. average LCP length.** Some building times are invisible because they are too small.




$n = 10^8$ $k = 10^5$

**Figure 7 Time-space trade-off in BFS-B-Hash.** The space shown is the memory required for the prefix tables. The dotted line is the input size. Note that by setting $b = 1$, the algorithm is equivalent to BFS-Hash.

the constant is much larger and its memory access pattern is much worse than the two hash-based solutions. We note that in some cases, the query time of BFS-SA can still be faster than BFS-Hash and BFS-B-Hash, especially when $k$ is large, which is consistent with the theory ($O(1)$ vs. $O(\log n)$ or $O(b \log n)$ per LCP query).

In theory, BFS-B-Hash reduces space usage in BFS-Hash by increasing the query time. Interestingly, when $k$ is small, BFS-B-Hash can also be faster than BFS-Hash by up to 2.5×. This is because BFS-B-Hash incurs fewer writes (and thus smaller memory footprints) in preprocessing that leads to faster building time. When $k$ is small, the running time is mostly dominated by the building time, and thus BFS-B-Hash can perform better. When $k$ is relatively large and $k^2$ is comparable to $n$, BFS-Hash becomes faster than BFS-B-Hash due to better LCP efficiency. In fact, when $k$ is large, the running time is mainly dominated by the query (BFS), and all three algorithms behave similarly. It is worth noting that in these experiments with $|\Sigma| = 256$ and random edits, in most of the cases, the queried LCP is small. Therefore, the $O(\log n)$ or $O(b \log n)$ query time for BFS-Hash and BFS-B-Hash are not tight, and they have much better memory access patterns than BFS-SA in LCP queries. As a result, they can have matching or even better performance than BFS-SA. Later we will show that under certain input distributions where the average LCP length is large, BFS-SA can have some advantage over both BFS-Hash and BFS-B-Hash.

**Real-World Datasets.** We now analyze how our algorithms perform on real-world string and edit patterns. The results are shown in the lower part of Fig. 5. The results are mostly consistent with our synthetic datasets, where BFS-B-Hash is more advantageous when $k$ is small, and BFS-Hash performs the best when $k$ is large. When $k$ is large, BFS-SA can also have comparable performance to the hash-based solutions.

**LCP Length vs. Performance.** It seems that for both synthetic and real-world data shown above, our hash-based solutions are always better than BFS-SA. It is worth asking, whether BFS-SA can give the best performance in certain cases, given that it has the best theoretical bounds (see Tab. 1). By investigating the bounds carefully, BFS-SA has better LCP query cost as $O(1)$, while the costs for BFS-Hash and BFS-B-Hash are $O(\log L)$ and $O(b \log L)$, respectively, where $L$ is the LCP length. This indicates that BFS-SA should be advantageous

**Table 3 Self-relative speedup of each implementation in each step.** "Build" = constructing the data structure for LCP queries. "Query" = the BFS process. "t.o." = timeout. We omit query speedup when $k = 10$ because there is little parallelism to be explored for BFS with small $k$, and the BFS time is also small and hardly affects the overall speedup. 192 hyperthreads are used for parallel executions.

| $n$ | $k$ | BFS-B-Hash | | | BFS-Hash | | | BFS-SA | | | DaC-SD |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Build | Query | Total | Build | Query | Total | Build | Query | Total | Total |
| $10^8$ | 10 | 20.4 | - | 19.9 | 46.6 | - | 46.5 | 49.6 | - | 49.4 | 68.2 |
| $10^9$ | $10^5$ | 24.2 | 36.4 | 36.3 | 42.7 | 46.8 | 46.6 | 51.2 | 27.1 | 48.3 | t.o. |

when $k$ and $L$ are both large. To verify this, we artificially created input instances with medium to large values of $k$ and controlled average LCP query lengths, and showed the results in Fig. 6 on two specific settings.

The experimental result is consistent with the theoretical analysis. The running time for BFS-Hash increases slowly with $L$, while the performance of BFS-B-Hash grows *much faster*, since it is affected by a factor of $O(b)$ more than BFS-Hash. The query time for BFS-SA almost stays the same, but also increases slightly with increasing $L$. This is because in general, with increasing $L$, the running time for all three algorithms may increase slightly due to worse cache locality in BFS due to more long matches. In Figure 6(a), the building time for both BFS-Hash and BFS-B-Hash are negligible, while BFS-SA still incurs significant building time. Even in this case, with an LCP length of 300, the query time of the hash-based solutions still becomes larger than the *total* running time of BFS-SA. In Figure 6(b) with a larger $k$, the building time for all three algorithms is negligible. In this case, BFS-SA always has comparable performance with BFS-Hash, and may perform better when $L > 20$. However, such extreme cases (both $k$ and $L$ are large) should be very rare in real-world datasets - when $k$ is large enough so that the query time is large enough to hide SA's building time, $L$ is more likely to be small, which in turn is beneficial for the query bounds in hash-based solutions. Indeed such cases did not appear in our 33 tests on both synthetic and real data.

**Parallelism.** We test the self-relative speedup of all algorithms. We present speedup numbers on two representative tests with different values of $n$ and $k$ in Tab. 3. For BFS-based algorithms, we separate the speedup for building and query. All our algorithms are highly parallelized. Even though BFS-SA and DaC-SD have a longer running time, they still have a 48–68× speedup, indicating good scalability. Our BFS-Hash algorithm has about 40–50× speedup in building, and BFS-B-Hash has a lower but decent speedup of about 20–40×. When $k$ is small, the frontier sizes (and the total work) of BFS are small, and the running time is also negligible. In this case, we cannot observe meaningful speedup. For larger $k = 10^5$, three BFS-based algorithms achieve 27–48× speedup both in query and entire edit distance algorithm.

**Space Usage.** We study the time-space tradeoff of our BFS-B-Hash with different block sizes $b$. We present the *auxiliary space* used by the prefix table in BFS-B-Hash along with running time in Fig. 7 using one test case with $n = 10^8$ and $k = 10^5$ in our synthetic dataset. The dotted line shows the input size. Note that when $b = 1$, it is exactly BFS-Hash. Since the inputs are 8-bit characters and the hash values are 64-bit integers, BFS-Hash incurs 8× space overhead than the input size. Using blocking, we can avoid such overhead and keep the auxiliary space even lower than the input. The auxiliary space decreases linearly with

the block size $b$. Interestingly, although blocking itself incurs time overhead, the impact in time is small: the time grows by 1.19× from $b = 1$ to 2, and grows by 1.08× from $b = 2$ to 64. This is mostly due to two reasons: 1) as mentioned, with 8-bit character input type and random edits, the average LCP length is likely short and within the first block, and therefore the query costs in both approaches are close to $O(L)$ for LCP length $L$, and 2) the extra factor of $b$ in queries (Line 17) is mostly cache hits (consecutive locations in an array). This illustrates the benefit of using blocking in such datasets, since blocking saves much space while only increasing the time by a small fraction.

## 7 Conclusion and Discussions

We proposed output-sensitive parallel algorithms for the edit-distance problem, as well as careful engineering of them. We revisited the BFS-based Landau-Vishkin algorithm. In addition to using SA as is used in Landau-Vishkin (our BFS-SA implementation), we also designed two hash-based data structures to replace the SA for more practical LCP queries (BFS-Hash and BFS-B-Hash). We also presented the first output-sensitive parallel algorithm based on divide-and-conquer with $\tilde{O}(nk)$ work and polylogarithmic span. We have also shown the best of our engineering effort on this algorithm, although its performance seems less competitive than other candidates due to work inefficiency.

We implemented all these algorithms and tested them on synthetic and real-world datasets. In summary, our BFS-based solutions show the best overall performance on datasets with real-world edits or random edits, due to faster preprocessing time and better I/O-friendliness. BFS-Hash performs the best in time when $k$ is large. BFS-B-Hash has better performance when $k$ is small. The blocking scheme also greatly improves space efficiency without introducing much overhead in time. In very extreme cases where both $k$ and the LCP lengths are large, BFS-SA can have some advantages over the hash-based solutions, while BFS-B-Hash can be much slower than BFS-Hash. However, such input patterns seem rare in the real world.

All our BFS-based solutions perform better than the output-insensitive solution in ParlayLib, and the DaC-based solution with $\tilde{O}(nk)$ work and polylogarithmic span, even for large $k > \sqrt{n}$. The results also imply the importance of work efficiency in parallel algorithm designs, consistent with the common belief in the literature [52, 24]. Because the number of cores in modern multi-core machines is small (usually hundreds to thousands) compared to the problem size, an algorithm is less practical if it blows up the work significantly, as parallelism cannot compensate for the performance loss due to larger work.

## References


1 Alberto Apostolico, Mikhail J Atallah, Lawrence L Larmore, and Scott McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM J. on Computing*, 19(5):968–988, 1990.

2 Nimar S Arora, Robert D Blumofe, and C Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. *Theory of Computing Systems (TOCS)*, 34(2):115–144, 2001.

3 K Nandan Babu and Sanjeev Saxena. Parallel algorithms for the longest common subsequence problem. In *IEEE International Conference on High Performance Computing (HiPC)*, pages 120–125. IEEE, 1997.

4 Michael A. Bender and Martin Farach-Colton. The lca problem revisited. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 88–94. Springer, 2000.

5 Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic acids research*, 41(D1):D36–D42, 2012.

6 Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Trans. on Comput.*, 38(11), 1989.

7 Guy E. Blelloch, Daniel Anderson, and Laxman Dhulipala. Parlaylib – A toolkit for parallel algorithms on shared-memory multicore machines. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 507–509, 2020.

8 Guy E. Blelloch, Jeremy T. Fineman, Yan Gu, and Yihan Sun. Optimal parallel algorithms in the binary-forking model. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 89–102, 2020.

9 Robert D. Blumofe and Charles E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM J. on Computing*, 27(1), 1998.

10 Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. Bad characters: Imperceptible nlp attacks. In *IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004. IEEE, 2022.

11 Dana Carroll. Focus: genome editing: genome editing: past, present, and future. *The Yale journal of biology and medicine*, 90(4):653, 2017.

12 Jung Hee Cheon, Miran Kim, and Kristin Lauter. Homomorphic computation of edit distance. In *International Conference on Financial Cryptography and Data Security*, pages 194–212. Springer, 2015.

13 Kendell Clement, Holly Rees, Matthew C Canver, Jason M Gehrke, Rick Farouni, Jonathan Y Hsu, Mitchel A Cole, David R Liu, J Keith Joung, Daniel E Bauer, et al. Crispresso2 provides accurate and rapid genome editing sequence analysis. *Nature biotechnology*, 37(3):224–226, 2019.

14 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3rd edition)*. MIT Press, 2009.

15 Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. McGraw-Hill Higher Education New York, 2008.

16 Xiangyun Ding, Xiaojun Dong, Yan Gu, Yihan Sun, and Youzhe Liu. Efficient parallel output-sensitive edit distance. *arXiv preprint:2306.17461*, 2023.

17 Xiangyun Ding, Xiaojun Dong, Yan Gu, Yihan Sun, and Youzhe Liu. Parallel implementations for output-sensitive edit distance. `https://github.com/ucrparlay/Edit-Distance`, 2023.

18 Robert W Floyd. Algorithm 97: shortest path. *Commun. ACM*, 5(6):345, 1962.

19 Zvi Galil and Raffaele Giancarlo. Improved string matching with $k$ mismatches. *ACM SIGACT News*, 17(4):52–54, 1986.

20 Zvi Galil and Raffaele Giancarlo. Parallel string matching with $k$ mismatches. *Theoretical Computer Science (TCS)*, 51(3):341–348, 1987.

21 Zvi Galil and Raffaele Giancarlo. Data structures and algorithms for approximate string matching. *Journal of Complexity*, 4(1):33–72, 1988.

22 Zvi Galil and Kunsoo Park. An improved algorithm for approximate string matching. *SIAM Journal on Computing*, 19(6):989–999, 1990.

23 Michael T Goodrich and Roberto Tamassia. *Algorithm design and applications*. Wiley Hoboken, 2015.

24 Yan Gu, Ziyang Men, Zheqi Shen, Yihan Sun, and Zijin Wan. Parallel longest increasing subsequence and van emde boas trees. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2023.

25 Yan Gu, Zachary Napier, and Yihan Sun. Analysis of work-stealing and parallel cache complexity. In *SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 46–60. SIAM, 2022.

26 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

27 Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

28 Daniel Hládek, Ján Staš, and Matúš Pleva. Survey of automatic spelling correction. *Electronics*, 9(10):1670, 2020.

29 Md Mosabbir Hossain, Md Farhan Labib, Ahmed Sady Rifat, Amit Kumar Das, and Monira Mukta. Auto-correction of english to bengali transliteration system using levenshtein distance. In *International Conference on Smart Computing & Communications (ICSCC)*, pages 1–5. IEEE, 2019.

30 Yoon-Seong Jeon, Kihyun Lee, Sang-Cheol Park, Bong-Soo Kim, Yong-Joon Cho, Sung-Min Ha, and Jongsik Chun. Ezeditor: a versatile sequence alignment editor for both rrna-and protein-coding genes. *International journal of systematic and evolutionary microbiology*, 64(Pt_2):689–691, 2014.

31 Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. *Journal of Computational Biology*, 9(2):371–388, 2002.

32 Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In *Intl. Colloq. on Automata, Languages and Programming (ICALP)*, pages 943–955. Springer, 2003.

33 Richard M Karp and Michael O Rabin. Efficient randomized pattern-matching algorithms. *IBM journal of research and development*, 31(2):249–260, 1987.

34 Peter Krusche and Alexander Tiskin. New algorithms for efficient parallel string comparison. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 209–216, 2010.

35 Gad M Landau and Uzi Vishkin. Efficient string matching with $k$ mismatches. *Theoretical Computer Science (TCS)*, 43:239–249, 1986.

36 Gad M Landau and Uzi Vishkin. Fast string matching with $k$ differences. *J. Computer and System Sciences*, 37(1):63–78, 1988.

37 Gad M Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989.

38 VI Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, page 707, 1966.

39 Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

40 Linux Kernel File dcn_1_0_sh_mask.h Commit History on GitHub. `https://github.com/torvalds/linux/blob/master/drivers/gpu/drm/amd/include/asic_reg/dcn/dcn_1_0_sh_mask.h`.

41 Mi Lu and Hua Lin. Parallel algorithms for the longest common subsequence problem. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):835–848, 1994.

42 Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM J. on Computing*, 22(5):935–948, 1993.

43 Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. Locality-sensitive hashing for the edit distance. *Bioinformatics*, 35(14):i127–i135, 2019.

44 Samuel McCauley. Approximate Similarity Search Under Edit Distance Using Locality-Sensitive Hashing. In *24th International Conference on Database Theory (ICDT 2021)*, volume 186, pages 21:1–21:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

45 Municipal history of Quebec on Wikipedia. `https://en.wikipedia.org/wiki/Municipal_history_of_Quebec`.

46 Eugene W Myers. An o (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

47 Eugene Wimberly Myers. *Incremental alignment algorithms and their applications*. University of Arizona, Department of Computer Science, 1986.

48 Jean-Frédéric Myoupo and David Seme. Time-efficient parallel algorithms for the longest common subsequence and related problems. *J. Parallel Distrib. Comput.*, 57(2):212–223, 1999.

49 Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

**50** Mike Paterson and Vlado Dančík. Longest common subsequences. In *International Symposium on Mathematical Foundations of Computer Science*, pages 127–142. Springer, 1994.

**51** Jane Peterson, Susan Garges, Maria Giovanni, Pamela McInnes, Lu Wang, Jeffery A Schloss, Vivien Bonazzi, Jean E McEwen, Kris A Wetterstrand, Carolyn Deal, et al. The nih human microbiome project. *Genome research*, 19(12):2317–2323, 2009.

**52** Zheqi Shen, Zijin Wan, Yan Gu, and Yihan Sun. Many sequential iterative algorithms can be parallel and (nearly) work-efficient. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2022.

**53** Julian Shun. Fast parallel computation of longest common prefixes. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, pages 387–398. IEEE, 2014.

**54** Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.

**55** Vianney Kengne Tchendji, Armel Nkonjoh Ngomade, Jerry Lacmou Zeutouo, and Jean Frédéric Myoupo. Efficient cgm-based parallel algorithms for the longest common subsequence problem with multiple substring-exclusion constraints. *Parallel Computing*, 91:102598, 2020.

**56** Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985.

**57** Liang-Jiao Xue and Chung-Jui Tsai. Ageseq: analysis of genome editing by sequencing. *Molecular plant*, 8(9):1428–1430, 2015.

**58** Jiaoyun Yang, Yun Xu, and Yi Shang. An efficient parallel algorithm for longest common subsequence problem on GPUs. In *World Congress on Engineering*, volume 1, pages 499–504, 2010.

**59** Hongyu Zhang. Alignment of blast high-scoring segment pairs based on the longest increasing subsequence algorithm. *Bioinformatics*, 19(11):1391–1396, 2003.

# Efficient 1-Laplacian Solvers for Well-Shaped Simplicial Complexes: Beyond Betti Numbers and Collapsing Sequences

**Ming Ding**
ETH Zürich, Switzerland

**Peng Zhang**
Rutgers University, New Brunswick, NJ, USA

## Abstract

We present efficient algorithms for solving systems of linear equations in 1-Laplacians of well-shaped simplicial complexes. 1-Laplacians, or higher-dimensional Laplacians, generalize graph Laplacians to higher-dimensional simplicial complexes and play a key role in computational topology and topological data analysis. Previously, nearly-linear time solvers were developed for simplicial complexes with known collapsing sequences and bounded Betti numbers, such as those triangulating a three-ball in $\mathbb{R}^3$ (Cohen, Fasy, Miller, Nayyeri, Peng, and Walkington [SODA'2014], Black, Maxwell, Nayyeri, and Winkelman [SODA'2022], Black and Nayyeri [ICALP'2022]). Furthermore, Nested Dissection provides quadratic time solvers for more general systems with nonzero structures representing well-shaped simplicial complexes embedded in $\mathbb{R}^3$.

We generalize the specialized solvers for 1-Laplacians to simplicial complexes with additional geometric structures but *without collapsing sequences and bounded Betti numbers*, and we improve the runtime of Nested Dissection. We focus on simplicial complexes that meet two conditions: (1) each individual simplex has a bounded aspect ratio, and (2) they can be divided into "disjoint" and balanced regions with well-shaped interiors and boundaries. Our solvers draw inspiration from the Incomplete Nested Dissection for stiffness matrices of well-shaped trusses (Kyng, Peng, Schwieterman, and Zhang [STOC'2018]).

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Mathematics of computing → Computations on matrices; Mathematics of computing → Algebraic topology

**Keywords and phrases** 1-Laplacian Solvers, Simplicial Complexes, Incomplete Nested Dissection

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.41

**Related Version** *Full Version*: `https://arxiv.org/abs/2302.06499` [20]

**Funding** This work was supported in part by NSF Grant CCF-2238682.

**Acknowledgements** We thank Rasmus Kyng for valuable discussions and the reviewers for their insightful comments.

## 1 Introduction

Combinatorial Laplacians generalize graph Laplacian matrices to higher dimensional simplicial complexes – a collection of 0-simplexes (vertices), 1-simplexes (edges), 2-simplexes (triangles), and their higher dimensional counterparts. Simplicial complexes encode higher-order relations between data points in a metric space. By studying the topological properties of these complexes using Combinatorial Laplacians, one can capture higher-order features that go beyond connectivity and clustering.


Given an oriented $d$-dimensional simplicial complex $\mathcal{K}$, for each $0 \leq i \leq d$, let $\mathcal{C}_i$ be the vector space generated by the $i$-simplexes in $\mathcal{K}$ with coefficients in $\mathbb{R}$. We can define a sequence of boundary operators:

$$\mathcal{C}_d \xrightarrow{\partial_d} \mathcal{C}_{d-1} \xrightarrow{\partial_{d-1}} \cdots \xrightarrow{\partial_2} \mathcal{C}_1 \xrightarrow{\partial_1} \mathcal{C}_0,$$

where each $\partial_i$ is a linear map that maps every $i$-simplex to a signed sum of its boundary $(i-1)$-faces. We define the $i$-Laplacian $\boldsymbol{L}_i : \mathcal{C}_i \to \mathcal{C}_i$ to be

$$\boldsymbol{L}_i = \partial_{i+1}\partial_{i+1}^\top + \partial_i^\top \partial_i. \tag{1}$$

In particular, $\partial_1$ is the vertex-edge incidence matrix, and $\boldsymbol{L}_0$ is the graph Laplacian (following the convention, we define $\partial_0 = \mathbf{0}$). One can assign weights to each simplex in $\mathcal{K}$ and define weighted Laplacians.

It is well-known that linear equations in graph Laplacians can be approximately solved in nearly-linear time in the number of nonzeros of the system [50, 30, 31, 29, 36, 46, 16, 34, 32, 27]. These fast Laplacian solvers have led to significant developments in algorithm design for graph problems such as maximum flow [41, 42, 10], minimum cost flow and lossy flow [37, 18], and graph sparsification [49], known as "the Laplacian Paradigm" [52].

Inspired by the success of graph Laplacians, Cohen, Fasy, Miller, Nayyeri, Peng, and Walkington [13] initiated the study of fast solvers for 1-Laplacian linear equations. They designed a nearly-linear time solver for simplicial complexes *with zero Betti numbers*[1] *and known collapsing sequences*. Later, Black, Maxwell, Nayyeri and Winkelman [5], and Black and Nayyeri [6] generalized this algorithm to subcomplexes of such a complex *with bounded first Betti numbers*[2]. One concrete example studied in these papers is convex simplicial complexes that piecewise linearly triangulate a convex ball in $\mathbb{R}^3$, for which a collapsing sequence exists and can be computed in linear time [11, 12]. However, deciding whether a simplicial complex has a collapsing sequence is NP-hard in general [51]; computing the Betti numbers is as hard as computing the ranks of general $\{0,1\}$ matrices [23]. In addition, 1-Laplacian systems for general simplicial complexes embedded in $\mathbb{R}^4$ are as hard to solve as general sparse linear equations [19], for which the best-known algorithms need super-quadratic time [47, 44]. All the above motivates the following question:

> Can we efficiently solve 1-Laplacian systems for other classes of *structured* simplicial complexes, e.g., *without known collapsing sequences and with arbitrary Betti numbers*?

In addition to the specialized solvers for 1-Laplacian systems mentioned above, Nested Dissection can solve 1-Laplacian systems in quadratic time for simplicial complexes in $\mathbb{R}^3$ with additional geometric structures [25, 39, 43] such as bounded aspect ratios[3] of individual tetrahedrons. Furthermore, iterative methods such as Preconditioned Conjugate Gradient approximately solve 1-Laplacian systems in time $\widetilde{O}(n\sqrt{\kappa})$, where $n$ is the number of simplexes and $\kappa$ is the condition number of the coefficient matrix.

Inspired by solvers that leverage geometric structures and spectral properties, we develop efficient 1-Laplacian solvers for well-shaped simplicial complexes embedded in $\mathbb{R}^3$ *without known collapsing sequences and with arbitrary Betti numbers*. Our solver adapts the Incomplete Nested Dissection algorithm, proposed by Kyng, Peng, Schwieterman, and Zhang [33]

---

[1] Informally, the $i$th Betti number is the number of $i$-dimensional holes on a topological surface. For example, the zeroth, first, and second Betti numbers represent the numbers of connected components, one-dimensional "circular" holes, and two-dimensional "voids" or "cavities," respectively.

[2] The solver has cubic dependence on the first Betti number.

[3] The aspect ratio of a geometric shape $S$ is the radius of the smallest ball containing $S$ divided by the radius of the largest ball contained in $S$.

for solving linear equations in well-shaped 3-dimensional truss stiffness matrices. These matrices represent another generalization of graph Laplacians; however, they differ quite from the 1-Laplacians studied in this paper. A primary distinction is that the kernel of a truss stiffness matrix has an explicit and well-understood form, while computing a 1-Laplacian's kernel is as hard as that for a general matrix.

## 1.1 Our Results

We say a simplex is *stable* if it has $O(1)$ aspect ratio and $\Theta(1)$ weight. We focus on a *pure* simplicial complex[4] $\mathcal{K}$ embedded in $\mathbb{R}^3$. We require $\mathcal{K}$ admits a nice division parameterized by $r \in \mathbb{R}_+$, called *$r$-hollowing*. We adopt and adapt the concept of $r$-hollowing introduced in [33] to suit our 1-Laplacian solvers. Informally, our $r$-hollowing for a simplicial complex containing $n$ simplexes divides $\mathcal{K}$ into $O(n/r)$ "separated" regions where each region has $O(r)$ simplexes and $O(r^{2/3})$ boundary simplexes. Only boundary simplexes can appear in multiple regions. Additionally, we mandate that each region's boundary triangulates a spherical shell in $\mathbb{R}^3$, exhibiting a *"hop" diameter* of $O(r^{1/3})$ and a *"hop" shell width* of at least 5. The formal definition is given in Definition 2.9. The bounded aspect ratio of each tetrahedron allows us to employ Nested Dissection for the interior simplexes within every region. The boundary shape requirement facilitates preconditioning the sub-system, derived from partial Nested Dissection, by the boundaries themselves and solving this sub-system using Preconditioned Conjugate Gradient.

Below, we present our main results informally. Firstly, we assume that an $r$-hollowing of a pure 3-complex is provided, which offers the broadest applicability of our algorithm. This assumption is justifiable when one can determine the construction of the simplicial complex; for instance, one can decide how to discretize a continuous topological space or how to triangulate a space given a set of points. Subsequently, we establish sufficient conditions for 3-complexes that allow us to compute $r$-hollowings in linear time.

▶ **Theorem 1.1** (Informal statement). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes. Given an $r$-hollowing for $\mathcal{K}$, for any $\epsilon > 0$, we can approximately solve a system in the 1-Laplacian of $\mathcal{K}$ within error $\epsilon$ in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2 r^{-2/3}\right)$. The runtime is $o(n^2)$ if $r = o(n)$ and $r = \omega(1)$. In particular, when $r = \Theta(n^{3/5})$, the runtime is minimized (up to constant) and equals $O(n^{8/5}\log(n/\epsilon))$.*

Our runtime in Theorem 1.1 does not depend on the Betti numbers of $\mathcal{K}$ and does not require collapsing sequences. When $r = o(n)$ and $r = \omega(1)$, the runtime is $o(n^2)$, asymptotically faster than Nested Dissection [43]. The solver in [6] for a 1-Laplacian system for the $\mathcal{K}$ stated in Theorem 1.1 is $\widetilde{O}(\beta^3 m)$[5], where $m$ is the number of simplexes in $\mathcal{X} \supset \mathcal{K}$ with a known collapsing sequence and $\beta$ is the first Betti number of $\mathcal{K}$. In the worst-case scenario, $m$ can be as large as $\Omega(n^2)$. But [6] does not require a known $r$-hollowing.

Without assuming prior knowledge about $r$-hollowing, the following theorem presents a solver with the same runtime as Theorem 1.1 when the 3-complex $\mathcal{K}$ satisfies additional geometric restrictions: First, the convex hull of $\mathcal{K}$ has $O(1)$ aspect ratio, and each tetrahedron of $\mathcal{K}$ has $\Theta(1)$ volume. Second, all but one the boundary components of $\mathcal{K}$, which correspond to "holes inside" $\mathcal{K}$, satisfy the following conditions: (1) every boundary component of $\mathcal{K}$ has

[4] A simplicial complex is *pure* if every maximal simplex (i.e., a simplex that is not a proper subset of any other simplex in the complex) has the same dimension. For example, a pure 3-complex is a tetrahedron mesh that consists of tetrahedrons and their sub-simplexes.

[5] We use $\widetilde{O}(\cdot)$ to hide polylog factors on the number of simplexes and the inverse of error parameter.

1-skeleton diameter $O(r^{1/3})$; (2) the total size of boundary components within any $\mathbb{X} \subset \mathbb{R}^3$ of volume $r$ is at most $O(r^{2/3})$, and the total size of boundary components of $\mathcal{K}$ is $O(nr^{-1/3})$; (3) the triangle distance between any two boundary components of $\mathcal{K}$ is greater than 5. These geometric conditions allow us to find an $r$-hollowing of $\mathcal{K}$ in linear time.

▶ **Theorem 1.2** (Informal statement). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes; assume $\mathcal{K}$ satisfies the aforementioned additional geometric structures with parameter $r$. Then, for any $\epsilon > 0$, we can approximately solve a system in the 1-Laplacian of $\mathcal{K}$ within error $\epsilon$ in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2 r^{-2/3}\right)$. In particular, when $r = \Theta(n^{3/5})$, the runtime is minimized (up to constant) and equals $O(n^{8/5}\log(n/\epsilon))$.*

We then examine unions of pure 3-complexes glued together by identifying certain subsets of simplexes on the boundary components (called exterior simplexes) of 3-complex chunks. Moreover, each 3-complex chunk admits a $\Theta(n_i^{3/5})$-hollowing with $n_i$ being the number of simplexes in this chunk. We remark that such a union of 3-complexes, called $\mathcal{U}$, *may not be embeddable in* $\mathbb{R}^3$. So, the previously established methods from [13, 5, 6] and Nested Dissection are unsuitable for this scenario. Building on our algorithm for Theorem 1.1, we design an efficient algorithm for $\mathcal{U}$ whose runtime depends sub-quadratically on the size of $\mathcal{U}$ and polynomially on the number of chunks and the number of simplexes shared by more than one chunk.

▶ **Theorem 1.3** (Informal statement). *Let $\mathcal{U}$ be a union of $h$ pure 3-complexes that are glued together by identifying certain subsets of their exterior simplexes. Each 3-complex chunk is embedded in $\mathbb{R}^3$, contains $n_i$ stable simplexes, and has a known $\Theta(n_i^{3/5})$-hollowing. For any $\epsilon > 0$, we can solve a system in the 1-Laplacian of $\mathcal{U}$ within error $\epsilon$ in time $\widetilde{O}(n^{8/5}k + h^2k^2 + k^3)$ where $n$ is the number of simplexes in $\mathcal{U}$, $k$ is the number of exterior simplexes shared by more than one complex chunk.*

When $h = \widetilde{O}(1)$ and $k = \widetilde{O}(n^{1/2})$, the solver in Theorem 1.3 has the same runtime as Theorem 1.1. When $h = o(n^{2/5}), k = o(n^{3/5})$, the runtime is $o(n^2)$, asymptotically faster than Nested Dissection.

## 1.2 Motivations and Applications

In the past decade, combinatorial Laplacians have played a crucial role in the development of computational topology and topological data analysis in various domains, such as statistics [28, 45], graphics and imaging [40, 53], brain networks [35], deep learning [8], signal processing [3], and cryo-electron microscope [54]. We recommend readers consult accessible surveys [26, 9, 22, 38] for more information.

Combinatorial Laplacians have their roots in the study of discrete Hodge decomposition [21], which states that the kernel of the $i$-Laplacian $\boldsymbol{L}_i$ is isomorphic to the $i$th homology group of the simplicial complex. Among the many applications of combinatorial Laplacians, a central problem is determining the Betti numbers – the ranks of the homology groups – which are important topological invariants. Additionally, discrete Hodge decomposition allows for the extraction of meaningful information from data by decomposing them into three mutually orthogonal components: gradient (in the image of $\partial_i^\top$), curl (in the image of $\partial_{i+1}$), and harmonic (in the kernel of $\boldsymbol{L}_i$) components. For instance, the three components of edge flows in a graph capture the global trends, local circulations, and "noise".

The computation of both Betti numbers and discrete Hodge decomposition of higher-order flows can be achieved by solving systems of linear equations in combinatorial Laplacians [24, 38]. The rank of a matrix $\boldsymbol{L}_i$ can be determined by solving a logarithmic number of linear

equation systems in $\boldsymbol{L}_i$ [2]. The discrete Hodge decomposition can be calculated by solving least square problems involving boundary operators or combinatorial Laplacians, which in turn reduces to solving linear equations in these matrices.

Furthermore, an important question in numerical linear algebra concerns whether the nearly-linear time solvers for graph Laplacian linear equations can be generalized to larger classes of linear equations. Researchers have achieved success with elliptic finite element systems [7], Connection Laplacians [32], directed Laplacians [15, 14], well-shaped truss stiffness matrices [17, 48, 33]. It would be intriguing to determine what structures of linear equations facilitate faster solvers. Another theoretically compelling reason for developing efficient solvers for 1-Laplacians stems from the "equivalence" of time complexity between solving 1-Laplacian systems and general sparse systems of linear equations [19]. If one can solve all 1-Laplacian systems in time $\widetilde{O}((\#\text{ of simplexes})^c)$ where $c \geq 1$ is a constant, then one can solve all general systems of linear equations in time $\widetilde{O}((\#\text{ of nonzero coefficients})^c)$.

# 2 Preliminaries

## 2.1 Background of Linear Algebra

Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, for $1 \leq i \leq n$, we let $\boldsymbol{x}[i]$ be the $i$th entry of $\boldsymbol{x}$; for $1 \leq i < j \leq n$, let $\boldsymbol{x}[i:j]$ be $(\boldsymbol{x}[i], \boldsymbol{x}[i+1], \ldots, \boldsymbol{x}[j])^\top$. The Euclidean norm of $\boldsymbol{x}$ is $\|\boldsymbol{x}\|_2 \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^n \boldsymbol{x}[i]^2}$. Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, for $1 \leq i \leq m, 1 \leq j \leq n$, we let $\boldsymbol{A}[i,j]$ be the $(i,j)$th entry of $\boldsymbol{A}$; for $S_1 \subseteq \{1, \ldots, m\}, S_2 \subseteq \{1, \ldots, n\}$, let $\boldsymbol{A}[S_1, S_2]$ be the submatrix with row indices in $S_1$ and column indices in $S_2$. Furthermore, we let $\boldsymbol{A}[S_1, :] = \boldsymbol{A}[S_1, \{1, \ldots, n\}]$ and $\boldsymbol{A}[:, S_2] = \boldsymbol{A}[\{1, \ldots, m\}, S_2]$. The operator norm of $\boldsymbol{A}$ (induced by the Euclidean norm) is $\|\boldsymbol{A}\|_2 \stackrel{\text{def}}{=} \max_{\boldsymbol{v} \in \mathbb{R}^n} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}$. The image of $\boldsymbol{A}$ is the linear span of the columns of $\boldsymbol{A}$, denoted by $\mathrm{Im}(\boldsymbol{A})$, and the kernel of $\boldsymbol{A}$ to be $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\}$, denoted by $\mathrm{Ker}(\boldsymbol{A})$. A fundamental theorem of Linear Algebra states $\mathbb{R}^m = \mathrm{Im}(\boldsymbol{A}) \oplus \mathrm{Ker}(\boldsymbol{A}^\top)$.

▶ **Fact 2.1.** [6] *For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $Im(\boldsymbol{A}) = Im(\boldsymbol{A}\boldsymbol{A}^\top)$.*

### Pseudo-inverse and Projection Matrix

The pseudo-inverse of $\boldsymbol{A}$ is defined to be a matrix $\boldsymbol{A}^\dagger$ that satisfies all the following four criteria: (1) $\boldsymbol{A}\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{A}$, (2) $\boldsymbol{A}^\dagger\boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger$, (3) $(\boldsymbol{A}\boldsymbol{A}^\dagger)^\top = \boldsymbol{A}\boldsymbol{A}^\dagger$, (4) $(\boldsymbol{A}^\dagger\boldsymbol{A})^\top = \boldsymbol{A}^\dagger\boldsymbol{A}$. The orthogonal projection matrix onto $\mathrm{Im}(\boldsymbol{A})$ is $\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} = \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^\dagger\boldsymbol{A}^\top$.

### Eigenvalues and Condition Numbers

Given a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, let $\lambda_{\max}(\boldsymbol{A})$ be the maximum eigenvalue of $\boldsymbol{A}$ and $\lambda_{\min}(\boldsymbol{A})$ the minimum *nonzero* eigenvalue of $\boldsymbol{A}$. The condition number of $\boldsymbol{A}$, denoted by $\kappa(\boldsymbol{A})$, is the ratio between $\lambda_{\max}(\boldsymbol{A})$ and $\lambda_{\min}(\boldsymbol{A})$. A symmetric matrix $\boldsymbol{A}$ is *positive semi-definite (PSD)* if all eigenvalues of $\boldsymbol{A}$ are non-negative. Let $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ be another square matrix. We say $\boldsymbol{A} \succcurlyeq \boldsymbol{B}$ if $\boldsymbol{A} - \boldsymbol{B}$ is PSD. The *condition number of $\boldsymbol{A}$ relative to $\boldsymbol{B}$* is

$$\kappa(\boldsymbol{A}, \boldsymbol{B}) \stackrel{\text{def}}{=} \min\left\{\frac{\alpha}{\beta} : \beta\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})}\boldsymbol{B}\boldsymbol{\Pi}_{\mathrm{Im}(\boldsymbol{A})} \preccurlyeq \boldsymbol{A} \preccurlyeq \alpha\boldsymbol{B}\right\}.$$

[6] All the facts in this section are well-known. For completeness, we include their proofs in the Appendix of the full paper [20].

▶ **Fact 2.2.** *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ be symmetric matrices such that $\boldsymbol{A} \preccurlyeq \boldsymbol{B}$. Then, for any $\boldsymbol{V} \in \mathbb{R}^{m\times n}$, $\boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^\top \preccurlyeq \boldsymbol{V}\boldsymbol{B}\boldsymbol{V}^\top$.*

### Schur Complement

Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, and let $F \cup C$ be a partition of $\{1, \ldots, n\}$. We write $\boldsymbol{A}$ as a block matrix:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}[F,F] & \boldsymbol{A}[F,C] \\ \boldsymbol{A}[C,F] & \boldsymbol{A}[C,C] \end{pmatrix}. \tag{2}$$

We define the (generalized) Schur complement of $\boldsymbol{A}$ onto $C$ to be

$$\mathrm{Sc}[\boldsymbol{A}]_C = \boldsymbol{A}[C,C] - \boldsymbol{A}[C,F]\boldsymbol{A}[F,F]^\dagger \boldsymbol{A}[F,C].$$

The Schur complement appears in performing a block Gaussian elimination on matrix $\boldsymbol{A}$ to eliminate the indices in $F$.

▶ **Fact 2.3.** *Let $\boldsymbol{A}$ be a PSD matrix defined in Equation* (2)*. Then,*

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{I} & \\ \boldsymbol{A}[C,F]\boldsymbol{A}[F,F]^\dagger & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{A}[F,F] & \\ & Sc[\boldsymbol{A}]_C \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{A}[F,F]^\dagger \boldsymbol{A}[F,C] \\ & \boldsymbol{I} \end{pmatrix}.$$

▶ **Fact 2.4.** *Let $\boldsymbol{A}$ be a PSD matrix defined in Equation* (2)*. Let $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^\top$, and we decompose $\boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}_F \\ \boldsymbol{B}_C \end{pmatrix}$ accordingly. Then, $Sc[\boldsymbol{A}]_C = \boldsymbol{B}_C \boldsymbol{\Pi}_{Ker(\boldsymbol{B}_F)} \boldsymbol{B}_C^\top$, where $\boldsymbol{\Pi}_{Ker(\boldsymbol{B}_F)}$ is the projection onto the kernel of $\boldsymbol{B}_F$.*

### Solving Linear Equations

We will need Fact 2.5 for relations between different error notations for linear equations and Theorem 2.6 for Preconditioned Conjugate Gradient.

▶ **Fact 2.5.** *Let $\boldsymbol{A}, \boldsymbol{Z} \in \mathbb{R}^{n\times n}$ be two symmetric PSD matrices, and let $\boldsymbol{\Pi}$ be the orthogonal projection onto $Im(\boldsymbol{A})$.*
1. *If $(1-\epsilon)\boldsymbol{A}^\dagger \preccurlyeq \boldsymbol{Z} \preccurlyeq (1+\epsilon)\boldsymbol{A}^\dagger$, then $\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{b} - \boldsymbol{b}\|_2 \le \epsilon\sqrt{\kappa(\boldsymbol{A})}\,\|\boldsymbol{b}\|_2$ for any $\boldsymbol{b} \in Im(\boldsymbol{A})$.*
2. *If $\|\boldsymbol{A}\boldsymbol{Z}\boldsymbol{b} - \boldsymbol{b}\|_2 \le \epsilon\,\|\boldsymbol{b}\|_2$ for any $\boldsymbol{b} \in Im(\boldsymbol{A})$, then $(1-\epsilon)\boldsymbol{A}^\dagger \preccurlyeq \boldsymbol{\Pi}\boldsymbol{Z}\boldsymbol{\Pi} \preccurlyeq (1+\epsilon)\boldsymbol{A}^\dagger$.*

▶ **Theorem 2.6** (Preconditioned Conjugate Gradient [1])**.** *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{n\times n}$ be two symmetric PSD matrices, and let $\boldsymbol{b} \in \mathbb{R}^n$. Each iteration of Preconditioned Conjugate Gradient multiplies one vector with $\boldsymbol{A}$, solves one system of linear equations in $\boldsymbol{B}$, and performs a constant number of vector operations. For any $\epsilon > 0$, the algorithm outputs an $\boldsymbol{x}$ satisfying $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2 \le \epsilon\,\|\boldsymbol{\Pi}_{\boldsymbol{A}}\boldsymbol{b}\|_2$ in $O(\sqrt{\kappa}\log(\kappa/\epsilon))$ such iterations, where $\boldsymbol{\Pi}_{\boldsymbol{A}}$ is the orthogonal projection matrix onto the image of $\boldsymbol{A}$ and $\kappa = \kappa(\boldsymbol{A}, \boldsymbol{B})$.*

## 2.2 Background of Topology

### Simplex and Simplicial Complexes

We consider a *d-simplex* (or $d$-dimensional simplex) $\sigma$ as an ordered set of $d+1$ vertices, denoted by $\sigma = [v_0, \ldots, v_d]$. A *face* of $\sigma$ is a simplex obtained by removing a subset of vertices from $\sigma$. A *simplicial complex* $\mathcal{K}$ is a finite collection of simplexes such that (1) for every $\sigma \in \mathcal{K}$ if $\tau \subset \sigma$ then $\tau \in \mathcal{K}$, and (2) for every $\sigma_1, \sigma_2 \in \mathcal{K}$, $\sigma_1 \cap \sigma_2$ is either empty or

a face of both $\sigma_1, \sigma_2$. The *dimension* of $\mathcal{K}$ is the maximum dimension of any simplex in $\mathcal{K}$. A *d-complex* is a $d$-dimensional simplicial complex. For $1 \leq i \leq d$, the *i-skeleton* of a $d$-complex $\mathcal{K}$ is the subcomplex consisting of all the simplexes of $\mathcal{K}$ of dimensions at most $i$. In particular, the 1-skeleton of $\mathcal{K}$ is a graph.

A *piecewise linear embedding* of a 3-complex in $\mathbb{R}^3$ maps a 0-simplex to a point, a 1-simplex to a line segment, a 2-simplex to a triangle, and a 3-simplex to a tetrahedron. In addition, the interior of the images of simplices are disjoint and the boundary of each simplex is mapped to the appropriate simplices. Such an embedding of a simplicial complex $\mathcal{K}$ defines an *underlying topological space* $\mathbb{K}$ – the union of the images of all the simplexes of $\mathcal{K}$. We say $\mathcal{K}$ is *convex* if $\mathbb{K}$ is convex. We say $\mathcal{K}$ *triangulates* a topological space $\mathbb{X}$ if $\mathbb{K}$ is homeomorphic to $\mathbb{X}$. A simplex $\sigma$ of $\mathcal{K}$ is a *exterior* simplex if $\sigma$ is contained in the boundary of $\mathbb{K}$, and $\sigma$ is an *interior* simplex otherwise. A connected component of exterior simplexes is called a *boundary component* of $\mathcal{K}$.

The *aspect ratio* of a set $S \subset \mathbb{R}^3$ is the radius of the smallest ball containing $S$ divided by the radius of the largest ball contained in $S$. The aspect ratio of $S$ is always greater than or equal to 1. We say a simplex $\sigma$ is *stable* if it has $O(1)$ aspect ratio and $\Theta(1)$ weight. Miller and Thurston proved the following lemma. As a corollary, the numbers of the vertices, the edges, the triangles, and the tetrahedrons of a 3-complex $\mathcal{K}$ that is composed of stable tetrahedrons are all equal up to a constant factor.

▶ **Lemma 2.7** (Lemma 4.1 of [43]). *Let $\mathcal{K}$ be a 3-complex in $\mathbb{R}^3$ in which each tetrahedron has $O(1)$ aspect ratio. Then, each vertex of $\mathcal{K}$ is contained in at most $O(1)$ tetrahedrons.*

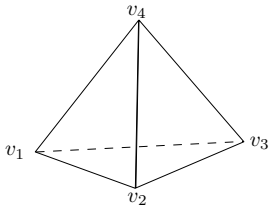
## Boundary Operators

An *i-chain* is a weighted sum of the oriented $i$-simplexes in $\mathcal{K}$ with the coefficients in $\mathbb{R}$. Let $\mathcal{C}_i$ denote the $i$th chain space. The *boundary operator* is a linear map $\partial_i : \mathcal{C}_i \to \mathcal{C}_{i-1}$ such that for an oriented $i$-simplex $\sigma = [v_0, v_1, \ldots, v_i]$,

$$\partial_i(\sigma) = \sum_{j=0}^{i} (-1)^j [v_0, \ldots, \hat{v}_j, \ldots, v_i],$$

where $[v_0, \ldots, \hat{v}_j, \ldots, v_i]$ is the oriented $(i-1)$-simplex obtained by removing $v_j$ from $\sigma$.

The operator $\partial_i$ can be written as a matrix in $|\mathcal{C}_{i-1}| \times |\mathcal{C}_i|$ dimensions, where the $(r, l)$th entry of $\partial_i$ is $\pm 1$ if the $r$th $(i-1)$-simplex is a face of the $l$th $i$-simplex and 0 otherwise. See Figure 1 for an example.




$$\partial_2 = \begin{array}{c} \\ [v_1,v_2] \\ [v_1,v_3] \\ [v_1,v_4] \\ [v_2,v_3] \\ [v_2,v_4] \\ [v_3,v_4] \end{array} \begin{array}{c} \begin{array}{cccc} [v_1,v_4,v_2] & [v_2,v_4,v_3] & [v_1,v_3,v_4] & [v_1,v_2,v_3] \end{array} \\ \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \end{bmatrix} \end{array}$$

**Figure 1** An example of boundary operator. The left side is a 3-simplex (a tetrahedron) with vertices $v_1, v_2, v_3, v_4$. The right side is the corresponding second boundary operator $\partial_2$, where each column corresponds to an oriented 2-simplex (a triangle) and each row corresponds to an oriented 1-simplex (an edge).

An important property of boundary operators is $\partial_i \partial_{i+1} = \mathbf{0}$, which implies $\mathrm{Im}(\partial_{i+1}) \subseteq \mathrm{Ker}(\partial_i)$. So, we can define the quotient space $H_i = \mathrm{Ker}(\partial_i) \backslash \mathrm{Im}(\partial_{i+1})$, called the *ith homology space* of $\mathcal{K}$. The rank of $H_i$ is called the $i$th Betti number of $\mathcal{K}$. If the $i$th Betti number of $\mathcal{K}$ is 0, then $\mathrm{Im}(\partial_i^\top) \oplus \mathrm{Im}(\partial_{i+1}) = \mathbb{R}^{|\mathcal{C}_i|}$. The first and second Betti numbers of a triangulation of a three-ball are both 0.

### Hodge Decomposition and Combinatorial Laplacians

Combinatorial Laplacians arise from the discrete Hodge decomposition.

▶ **Theorem 2.8** (Hodge decomposition [38]). *Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$ be matrices satisfying $\boldsymbol{AB} = \mathbf{0}$. Then, there is an orthogonal direct sum decomposition*

$$\mathbb{R}^n = \mathit{Im}(\boldsymbol{A}^\top) \oplus \mathit{Ker}(\boldsymbol{A}^\top \boldsymbol{A} + \boldsymbol{B}\boldsymbol{B}^\top) \oplus \mathit{Im}(\boldsymbol{B}).$$

Since $\partial_i \partial_{i+1} = \mathbf{0}$, it is valid to set $\boldsymbol{A} = \partial_i$ and $\boldsymbol{B} = \partial_{i+1}$. The matrix we get in the middle term is the *combinatorial Laplacian*: $\boldsymbol{L}_i \stackrel{\text{def}}{=} \partial_i^\top \partial_i + \partial_{i+1} \partial_{i+1}^\top$.

The weighted combinatorial Laplacian generalizes combinatorial Laplacian. For each $1 \le i \le d$, we assign each $i$-simplex of $\mathcal{K}$ with a *positive* weight, and let $\boldsymbol{W}_i : \mathcal{C}_i \to \mathcal{C}_i$ be a diagonal matrix where $\boldsymbol{W}_i[\sigma, \sigma]$ is the weighted of the $i$-simplex $\sigma$. Then the weighted *$i$-Laplacian* of $\mathcal{K}$ is a linear operator $\boldsymbol{L}_i : \mathcal{C}_i \to \mathcal{C}_i$ defined as

$$\boldsymbol{L}_i \stackrel{\text{def}}{=} \partial_i^\top \boldsymbol{W}_{i-1} \partial_i + \partial_{i+1} \boldsymbol{W}_{i+1} \partial_{i+1}^\top.$$

Note that Hodge decomposition also applies to weighted combinatorial Laplacian (by setting $\boldsymbol{A} = \boldsymbol{W}_{i-1}^{1/2} \partial_i$ and $\boldsymbol{B} = \partial_{i+1} \boldsymbol{W}_{i+1}^{1/2}$, we have $\boldsymbol{AB} = \mathbf{0}$). We call $\boldsymbol{L}_i^{\text{down}} \stackrel{\text{def}}{=} \partial_i^\top \boldsymbol{W}_{i-1} \partial_i$ the $i$th *down-Laplacian* operator and $\boldsymbol{L}_i^{\text{up}} \stackrel{\text{def}}{=} \partial_{i+1} \boldsymbol{W}_{i+1} \partial_{i+1}^\top$ the $i$th *up-Laplacian* operator. Sometimes, we use subscripts to specify the complex on which these operators are defined: $\partial_{i,\mathcal{K}}$, $\boldsymbol{W}_{i,\mathcal{K}}$, $\boldsymbol{L}_{i,\mathcal{K}}^{\text{down}}$, $\boldsymbol{L}_{i,\mathcal{K}}^{\text{up}}$.

### $r$-Hollowings

Let $\mathcal{K}$ be a pure 3-complex with $n$ simplexes. A set of triangles $\hat{\triangle}_1, \ldots, \hat{\triangle}_k$ form a *triangle path* of length $k-1$ if for any $1 \le i \le k-1$, $\hat{\triangle}_i$ and $\hat{\triangle}_{i+1}$ share an edge. The *triangle distance* between two triangles $\triangle_1$ and $\triangle_2$ is the shortest triangle path length between $\triangle_1$ and $\triangle_2$. The *triangle diameter* of $\mathcal{K}$ is the longest triangle distance between any two triangles. A *spherical shell* is $\{\boldsymbol{x} \in \mathbb{R}^3 : R_1 \le \|\boldsymbol{x}\|_2 \le R_2\}$ where $R_1 < R_2$. If $\mathcal{K}$ triangulates a spherical shell, we define the *shell width* to be the shortest triangle distance between any two triangles where one is on the outer sphere and one is on the inner sphere.

▶ **Definition 2.9** ($r$-hollowing). *Let $\mathcal{K}$ be a 3-complex with $n$ simplexes, and let $r = o(n)$ be a positive number. We divide $\mathcal{K}$ into $O(n/r)$* regions *each of $O(r)$ simplexes and $O(r^{2/3})$ boundary simplexes. Only boundary simplexes can appear in more than one region. The boundary of each region triangulates a spherical shell in $\mathbb{R}^3$ and has triangle diameter $O(r^{1/3})$ and shell width at least 5. The union of all boundary simplexes of each region is referred to as an* $r$-hollowing *of $\mathcal{K}$.*

In addition, this paper also examines sufficient conditions for 3-complexes that enable us to compute an $r$-hollowing in linear time (Algorithm 2 and refer to Figure 2 for an illustration). Specifically, we consider a pure 3-complex $\mathcal{K}$ embedded in $\mathbb{R}^3$ with $n$ stable simplexes possessing the following additional geometric structures:

1. The aspect ratio of the convex hull of $\mathcal{K}$ is $O(1)$, and the volume of each tetrahedron in $\mathcal{K}$ is $\Theta(1)$.
2. All but one boundary component has 1-skeleton diameter $O(r^{1/3})$.
3. The total number of exterior simplexes of $\mathcal{K}$ within any $\mathbb{X} \subset \mathbb{R}^3$ of volume $r$ is $O(r^{2/3})$; the total number of exterior simplexes of $\mathcal{K}$ is $O(nr^{-1/3})$.
4. The triangle distance between any two boundary components of $\mathcal{K}$ is greater than 5.

It is worth noting that fulfilling the aforementioned assumptions is not excessively challenging. On one end of the spectrum, there are scenarios where $\mathcal{K}$ contains at most $O(n/r)$ 2-dimensional holes, each with an interior volume of $O(r)$. On the other end, there are instances where $\mathcal{K}$ encompasses $O(nr^{-1/3})$ uniformly distributed small holes, each with a constant interior volume. Moreover, it is likely that all scenarios lying between these extremes would also meet these assumptions.

## 3 Main Theorems

We formally state our main results as follows.

▶ **Theorem 3.1.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ consisting of $n$ stable simplexes and with a known $r$-hollowing. Let $\boldsymbol{L}_1$ be the 1-Laplacian operator of $\mathcal{K}$, and let $\boldsymbol{\Pi}_1$ be the orthogonal projection matrix onto the image of $\boldsymbol{L}_1$. For any vector $\boldsymbol{b}$ and $\epsilon > 0$, we can find a solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2$ in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2r^{-2/3}\right)$.*

We will overview our algorithm for Theorem 3.1 in Section 4 and prove in Section 5.

▶ **Theorem 3.2.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ consisting of $n$ stable simplexes. Suppose $\mathcal{K}$ satisfies the additional geometric structures 1-4 with parameter $r = o(n)$. Let $\boldsymbol{L}_1$ be the 1-Laplacian operator of $\mathcal{K}$, and let $\boldsymbol{\Pi}_1$ be the orthogonal projection matrix onto the image of $\boldsymbol{L}_1$. For any vector $\boldsymbol{b}$ and $\epsilon > 0$, we can find a solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2$ in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2r^{-2/3}\right)$.*

The known $r$-hollowing assumption is replaced with geometric structures in Theorem 3.2, and a linear time algorithm for finding $r$-hollowing is presented in Section 6. It is worth mentioning that the additional geometric structures are introduced to ensure the feasibility of finding an $r$-hollowing in linear time. However, the algorithm for solving the system of linear equations remains the same.

▶ **Theorem 3.3.** *Let $\mathcal{U}$ be a union of $h$ pure 3-complexes glued together by identifying certain subsets of their exterior simplexes. Each 3-complex chunk is embedded in $\mathbb{R}^3$ and comprises $n_i$ stable simplexes, and has a known $\Theta(n_i^{3/5})$-hollowing. Let $\boldsymbol{L}_1$ be the 1-Laplacian operator of $\mathcal{U}$, and let $\boldsymbol{\Pi}_1$ be the orthogonal projection matrix onto the image of $\boldsymbol{L}_1$. For any vector $\boldsymbol{b}$ and $\epsilon > 0$, we can find a solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1\tilde{\boldsymbol{x}} - \boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2$ in time $\widetilde{O}(n^{8/5}k + h^2k^2 + k^3)$, where $n$ is the number of simplexes in $\mathcal{U}$, $k$ is the number of exterior simplexes shared by more than one chunk.*

Due to space constraints, the proof of Theorem 3.3 can be found in the full version of the paper [20].

# 4 Algorithm Overview

Cohen, Fasy, Miller, Nayyeri, Peng, and Walkington [13] observed that

$$\boldsymbol{L}_1^{\dagger} = \left(\boldsymbol{L}_1^{\text{down}}\right)^{\dagger} + \left(\boldsymbol{L}_1^{\text{up}}\right)^{\dagger},$$

where $\boldsymbol{L}_1^{\text{down}} = \partial_1^{\top} \boldsymbol{W}_0 \partial_1$ is the down-Laplacian and $\boldsymbol{L}_1^{\text{up}} = \partial_2 \boldsymbol{W}_2 \partial_2^{\top}$ is the up-Laplacian. The orthogonal projection matrices onto $\text{Im}(\partial_1^{\top})$ and $\text{Im}(\partial_2)$ are:

$$\boldsymbol{\Pi}_1^{\text{down}} \stackrel{\text{def}}{=} \partial_1^{\top}(\partial_1 \partial_1^{\top})^{\dagger}\partial_1, \ \boldsymbol{\Pi}_1^{\text{up}} \stackrel{\text{def}}{=} \partial_2(\partial_2^{\top}\partial_2)^{\dagger}\partial_2^{\top}.$$

▶ **Lemma 4.1** (Lemma 4.1 of [13]). *Let $\boldsymbol{b}$ be a vector. Consider the systems of linear equations: $\boldsymbol{L}_1\boldsymbol{x} = \boldsymbol{\Pi}_1\boldsymbol{b}$, $\boldsymbol{L}_1^{up}\boldsymbol{x}^{up} = \boldsymbol{\Pi}_1^{up}\boldsymbol{b}$, $\boldsymbol{L}_1^{down}\boldsymbol{x}^{down} = \boldsymbol{\Pi}_1^{down}\boldsymbol{b}$. Then, $\boldsymbol{x} = \boldsymbol{\Pi}_1^{up}\boldsymbol{x}^{up} + \boldsymbol{\Pi}_1^{down}\boldsymbol{x}^{down}$.*

Lemma 4.1 implies that four operators are needed to approximate $\boldsymbol{L}_1^{\dagger}$: (1) an approximate projection operator $\widetilde{\boldsymbol{\Pi}}_1^{\text{down}} \approx \boldsymbol{\Pi}_1^{\text{down}}$, (2) an approximate projection operator $\widetilde{\boldsymbol{\Pi}}_1^{\text{up}} \approx \boldsymbol{\Pi}_1^{\text{up}}$, (3) a down-Laplacian solver $\boldsymbol{Z}_1^{\text{down}}$ such that $\boldsymbol{L}_1^{\text{down}}\boldsymbol{Z}_1^{\text{down}}\boldsymbol{b} \approx \boldsymbol{b}$ for any $\boldsymbol{b} \in \text{Im}(\boldsymbol{L}_1^{\text{up}})$, and (4) an up-Laplacian solver $\boldsymbol{Z}_1^{\text{up}}$ such that $\boldsymbol{L}_1^{\text{up}}\boldsymbol{Z}_1^{\text{up}}\boldsymbol{b} \approx \boldsymbol{b}$ for any $\boldsymbol{b} \in \text{Im}(\boldsymbol{L}_1^{\text{up}})$.

We will apply the same approximate orthogonal projection $\widetilde{\boldsymbol{\Pi}}_1^{\text{down}}$ given in [13], which does not depend on Betti numbers. Our solver for the down 1-Laplacian is a slight modification of the one in [13] to incorporate the simplex weights. We state the two lemmas below.

▶ **Lemma 4.2** (Down-projection operator, Lemma 3.2 of [13]). *Let $\mathcal{K}$ be a 3-complex with $n$ simplexes. For any $\epsilon > 0$, there exists a linear operator $\widetilde{\boldsymbol{\Pi}}_1^{down}$ such that*

$$(1-\epsilon)\boldsymbol{\Pi}_1^{down} \preccurlyeq \widetilde{\boldsymbol{\Pi}}_1^{down}(\epsilon) \preccurlyeq \boldsymbol{\Pi}_1^{down}.$$

▶ **Lemma 4.3** (Down-Laplacian solver). *Let $\mathcal{K}$ be a weighted simplicial complex, and let $\boldsymbol{b} \in Im(\boldsymbol{L}_1^{down})$. There exists an operator $\boldsymbol{Z}_1^{down}$ such that $\boldsymbol{L}_1^{down}\boldsymbol{Z}_1^{down}\boldsymbol{b} = \boldsymbol{b}$. In addition, we can compute $\boldsymbol{Z}_1^{down}\boldsymbol{b}$ in linear time.*

## 4.1 Solver for Up-Laplacian

One of our primary technical contributions is the development of an efficient solver for the up-Laplacian system, stated in Lemma 4.4. We will describe the key idea behind our solver in this section.

▶ **Lemma 4.4** (Up-Laplacian solver). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes. Suppose we are given an $r$-hollowing for $\mathcal{K}$. Then for any $\epsilon > 0$, there exists an operator $\boldsymbol{Z}_1^{up}$ such that*

$$\forall \boldsymbol{b} \in Im(\boldsymbol{L}_1^{up}), \ \left\|\boldsymbol{L}_1^{up}\boldsymbol{Z}_1^{up}\boldsymbol{b} - \boldsymbol{b}\right\|_2 \le \epsilon \left\|\boldsymbol{b}\right\|_2.$$

*In addition, $\boldsymbol{Z}_1^{up}\boldsymbol{b}$ can be computed in time $O\left(nr + n^{4/3}r^{5/18}\log(n/\epsilon) + n^2 r^{-2/3}\right)$.*

We remark that Lemma 4.4 can be improved to $\widetilde{O}(n^{3/2})$ by using a slightly different $r$-hollowing (proved in the full version [20]), which might be of independent interest. Since the bottleneck of our solver for 1-Laplacians is from the projection for up 1-Laplaicans, we use the same $r$-hollowing here.

The given $O(n^{3/5})$-hollowing suggests a partition of the edges in $\mathcal{K}$ into $F \cup C$. We will explain the concrete partition shortly. We have the following matrix identity:

$$\boldsymbol{L}_1^{\text{up}} = \begin{pmatrix} \boldsymbol{I} & \\ \boldsymbol{L}_1^{\text{up}}[C,F]\boldsymbol{L}_1^{\text{up}}[F,F]^{\dagger} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{L}_1^{\text{up}}[F,F] & \\ & \text{Sc}[\boldsymbol{L}_1^{\text{up}}]_C \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{L}_1^{\text{up}}[F,F]^{\dagger}\boldsymbol{L}_1^{\text{up}}[F,C] \\ & \boldsymbol{I} \end{pmatrix},$$

where

$$\mathrm{Sc}[\boldsymbol{L}_1^{\mathrm{up}}]_C = \boldsymbol{L}_1^{\mathrm{up}}[C,C] - \boldsymbol{L}_1^{\mathrm{up}}[C,F]\boldsymbol{L}_1^{\mathrm{up}}[F,F]^{\dagger}\boldsymbol{L}_1^{\mathrm{up}}[F,C].$$

The following Lemma 4.5 reduces (approximately) solving a system in $\boldsymbol{L}_1^{\mathrm{up}}$ to (approximately) solving two systems in $\boldsymbol{L}_1^{\mathrm{up}}[F,F]$ and one system in $\mathrm{Sc}[\boldsymbol{L}_1^{\mathrm{up}}]_C$, whose proof can be found in the Appendix of the full version [20]. It is worth noting that Lemma 4.5 holds if we replace $\boldsymbol{L}_1^{\mathrm{up}}$ with an arbitrary symmetric PSD matrix, and we will apply it or its variants for different PSD matrices in our solvers. To avoid introducing additional notations, we state the lemma below in terms of $\boldsymbol{L}_1^{\mathrm{up}}$.

▶ **Lemma 4.5.** *Suppose we have two operators (1)* UpLapFSolver$(\cdot)$ *such that given any* $\boldsymbol{b} \in Im(\boldsymbol{L}_1^{up}[F,F])$, UpLapFSolver$(\boldsymbol{b})$ *returns a vector* $\boldsymbol{x}$ *satisfying* $\boldsymbol{L}_1^{up}[F,F]\boldsymbol{x} = \boldsymbol{b}$, *and (2)* SchurSolver$(\cdot,\cdot)$ *such that for any* $\boldsymbol{h} \in Im(Sc[\boldsymbol{L}_1^{up}]_C)$ *and* $\delta > 0$, SchurSolver$(\boldsymbol{h},\delta)$ *returns* $\tilde{\boldsymbol{x}}$ *satisfying* $\|Sc[\boldsymbol{L}_1^{up}]_C\tilde{\boldsymbol{x}} - \boldsymbol{h}\|_2 \le \delta\|\boldsymbol{h}\|_2$. *Given any* $\boldsymbol{b} = \begin{pmatrix}\boldsymbol{b}_F\\ \boldsymbol{b}_C\end{pmatrix} \in Im(\boldsymbol{L}_1^{up})$ *and any* $\epsilon > 0$, *let*

$$\begin{aligned}
&\boldsymbol{h} = \boldsymbol{b}_C - \boldsymbol{L}_1^{up}[C,F]\cdot \textsc{UpLapFSolver}(\boldsymbol{b}_F),\\
&\tilde{\boldsymbol{x}}_C = \textsc{SchurSolver}(\boldsymbol{h},\delta),\\
&\tilde{\boldsymbol{x}}_F = \textsc{UpLapFSolver}\left(\boldsymbol{b}_F - \boldsymbol{L}_1^{up}[F,C]\tilde{\boldsymbol{x}}_C\right),
\end{aligned} \tag{3}$$

*where* $\delta \le \frac{\epsilon}{1+\left\|\boldsymbol{L}_1^{up}[C,F]\boldsymbol{L}_1^{up}[F,F]^{\dagger}\right\|_2}$. *Then,*

$$\|\boldsymbol{L}_1^{up}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \le \epsilon\|\boldsymbol{b}\|_2,$$

*where* $\tilde{\boldsymbol{x}} = \begin{pmatrix}\tilde{\boldsymbol{x}}_F\\ \tilde{\boldsymbol{x}}_C\end{pmatrix}$. *Let* $m_F = |F|$ *and* $m_C = |C|$, *and let* UpLapFSolver *have runtime* $t_1(m_F)$ *and* SchurSolver *have runtime* $t_2(m_C)$. *Then, we can compute* $\tilde{\boldsymbol{x}}$ *in time* $O(t_1(m_F) + t_2(m_C) + m_F + m_C)$.

### 4.1.1 Partitioning the Edges

As suggested by Lemma 4.5, we want to partition the edges of $\mathcal{K}$ into $F \cup C$ so that both systems in $\boldsymbol{L}_1^{\mathrm{up}}[F,F]$ and the Schur complement $\mathrm{Sc}[\boldsymbol{L}_1^{\mathrm{up}}]_C$ can be efficiently solved. The given $O(n^{3/5})$-hollowing divides $\mathcal{K}$ into *"disjoint" and balanced* regions with *small* boundary. Let $F$ be the set of the "interior" edges of the regions and $C$ be the set of the "boundary" edges.

We first show the interiors of different regions are "disjoint" in the sense that $\boldsymbol{L}_1^{\mathrm{up}}[F,F]$ is a block diagonal matrix where each diagonal block corresponds to the interior of a region. We can write $\boldsymbol{L}_1^{\mathrm{up}}$ as the sum of rank-1 matrices that each corresponds to a triangle in $\mathcal{K}$:

$$\boldsymbol{L}_1^{\mathrm{up}} = \partial_2 \boldsymbol{W}_2 \partial_2^{\top} = \sum_{\sigma:\text{triangle in } \mathcal{K}} \boldsymbol{W}_2[\sigma,\sigma]\cdot\partial_2[:,\sigma]\partial_2[:,\sigma]^{\top}. \tag{4}$$

For any two edges $e_1, e_2$, $\boldsymbol{L}_1^{\mathrm{up}}[e_1,e_2] = 0$ if and only if no triangle in $\mathcal{K}$ contains both $e_1, e_2$. By our definition of $r$-hollowing in Definition 2.9, for different regions $R_1, R_2$ of $\mathcal{K}$ w.r.t. an $r$-hollowing, no triangle contains both an edge from $R_1$ and an edge from $R_2$.

In addition, the following lemma shows that the boundaries of the regions well approximate the Schur complement onto the boundaries. We give a formal proof of Lemma 4.6 in the full version [20].

▶ **Lemma 4.6** (Spectral bounds for $r$-hollowing)**.** *Let* $\mathcal{K}$ *be a pure* 3*-complex embedded in* $\mathbb{R}^3$ *composed of stable simplexes. Let* $\mathcal{T}$ *be an* $r$*-hollowing of* $\mathcal{K}$, *and let* $C$ *be the edges of* $\mathcal{T}$. *Then,* $\boldsymbol{L}_{1,\mathcal{T}}^{up} \preccurlyeq Sc[\boldsymbol{L}_1^{up}]_C \preccurlyeq O(r)\boldsymbol{L}_{1,\mathcal{T}}^{up}$.

### 4.1.2 Proof of Lemma 4.4 for Up-Laplacian Solver

Algorithm 1 sketches a pseudo-code for our up-Laplacian solver.

**Algorithm 1** UpLapSolver$(\mathcal{K}, \mathcal{T}, \boldsymbol{b}, \epsilon)$.

**Input:** A pure 3-complex $\mathcal{K}$ of $n$ stable simplexes with up-Laplacian $\boldsymbol{L}_1^{\text{up}}$, an $O(n^{3/5})$-hollowing $\mathcal{T}$, a vector $\boldsymbol{b} \in \text{Im}(\boldsymbol{L}_1^{\text{up}})$, an error parameter $\epsilon > 0$

**Output:** An approximate solution $\tilde{\boldsymbol{x}}$ such that $\|\boldsymbol{L}_1^{\text{up}}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{b}\|_2$

1 $F \leftarrow$ the interior edges of regions of $\mathcal{K}$ w.r.t. $\mathcal{T}$, $C \leftarrow$ the boundary edges of regions.

2 UpLapFSolver$(\cdot) \leftarrow$ a solver by Nested Dissection that satisfies the requirement in Lemma 4.5.

3 SchurSolver$(\cdot,\cdot) \leftarrow$ a solver by Preconditioned Conjugate Gradient with the preconditioner being the up-Laplacian of $\mathcal{T}$ that satisfies the requirement in Lemma 4.5.

4 $\tilde{\boldsymbol{x}} \leftarrow$ computed by Equation (3)

5 **return** *solution* $\tilde{\boldsymbol{x}}$

By Lemma 4.5, the $\tilde{\boldsymbol{x}}$ returned by Algorithm 1 satisfies $\|\boldsymbol{L}_1^{\text{up}}\tilde{\boldsymbol{x}} - \boldsymbol{b}\|_2 \leq \epsilon \|\boldsymbol{b}\|_2$. To bound the runtime of Algorithm 1, we need the following lemmas for lines 2 and 3.

▶ **Lemma 4.7** (Solver for the "$F$" part)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes. Let $\mathcal{T}$ be an $r$-hollowing of $\mathcal{K}$, and let $F$ be the set of interior edges in each region of $\mathcal{K}$ w.r.t. $\mathcal{T}$. Then, with a pre-processing time $O(nr)$, there exists a solver UpLapFSolver$(\cdot)$ such that given any $\boldsymbol{b}_F \in im(\boldsymbol{L}_1^{up}[F,F])$, UpLapFSolver$(\boldsymbol{b}_F)$ returns an $\boldsymbol{x}_F$ such that $\boldsymbol{L}_1^{up}[F,F]\boldsymbol{x}_F = \boldsymbol{b}_F$ in time $O(nr^{1/3})$.*

By our choice of $F$, the matrix $\boldsymbol{L}_1^{\text{up}}[F,F]$ can be written as a block diagonal matrix where each block corresponds to a region of $\mathcal{K}$ w.r.t. the $r$-hollowing $\mathcal{T}$. Since each region is a 3-complex in which every tetrahedron has an aspect ratio $O(1)$, we can construct the solver UpLapFSolver by Nested Dissection [43]. However, since each row or column of $\boldsymbol{L}_1^{\text{up}}[F,F]$ corresponds to an edge, we need to turn the good *vertex separators* in [43] into good *edge separators* for regions of $\mathcal{K}$. The proof of Lemma 4.7 can be found in the full version [20].

▶ **Lemma 4.8** (Solver for the Schur complement)**.** *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes. Let $\mathcal{T}$ be an $r$-hollowing of $\mathcal{K}$, and let $C$ be the set of boundary edges of each region of $\mathcal{K}$ w.r.t. $\mathcal{T}$. Then, with a pre-processing time $O(nr + n^2r^{-2/3})$ there exists a solver SchurSolver$(\cdot,\cdot)$ such that for any $\boldsymbol{h} \in Im(Sc[\boldsymbol{L}_1^{up}]_C)$ and $\delta > 0$, SchurSolver$(\boldsymbol{h}, \delta)$ returns an $\tilde{\boldsymbol{x}}_C$ such that $\|Sc[\boldsymbol{L}_1^{up}]_C\tilde{\boldsymbol{x}}_C - \boldsymbol{h}\|_2 \leq \delta \|\boldsymbol{h}\|_2$ in time $\widetilde{O}\left(nr^{5/6} + n^{4/3}r^{5/18}\right)$.*

Our solver SchurSolver is based on the Preconditioned Conjugate Gradient (PCG) with the preconditioner $\boldsymbol{L}_{1,\mathcal{T}}^{\text{up}}$, the up-Laplacian operator of $\mathcal{T}$. By Theorem 2.6 and Lemma 4.6, the number of PCG iterations is $\widetilde{O}(\sqrt{r})$. In each PCG iteration, we solve the system in $\boldsymbol{L}_{1,\mathcal{T}}^{\text{up}}$ via Nested Dissection. Again, the proof of Lemma 4.8 can be found in the full version of the paper [20].

Given the above lemmas, we prove Lemma 4.4.

**Proof of Lemma 4.4.** The correctness of Algorithm 1 is by Lemma 4.5. By Lemma 4.7 and 4.8, the total runtime of the algorithm is

$$\widetilde{O}\left(nr^{5/6} + n^{4/3}r^{5/18} + nr + n^2r^{-2/3}\right). \qquad \blacktriangleleft$$

## 4.2 Projection for Up 1-Laplacian

As the first Betti number of $\mathcal{K}$ can be arbitrary, the approximate projection operators for the up 1-Laplacian provided in [13, 5, 6] are not applicable here. Our approximate projection operator follows a similar approach to our up 1-Laplacian solver, which is based on an incomplete Nested Dissection for *triangles*, instead of edges.

▶ **Lemma 4.9** (Up-projection operator). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes. Suppose we are given an $r$-hollowing for $\mathcal{K}$. Then, for any $\epsilon > 0$, there exists an operator $\widetilde{\mathbf{\Pi}}_1^{up}$ such that*

$$\forall \boldsymbol{b}, \left\| \widetilde{\mathbf{\Pi}}_1^{up} \boldsymbol{b} - \mathbf{\Pi}_1^{up} \boldsymbol{b} \right\|_2 \leq \epsilon \left\| \mathbf{\Pi}_1^{up} \boldsymbol{b} \right\|_2 .$$

*In addition, $\widetilde{\mathbf{\Pi}}_1^{up} \boldsymbol{b}$ can be computed in time $O\left(nr + n^{4/3} r^{5/18} \log(n/\epsilon) + n^2 r^{-2/3}\right)$.*

The proof of Lemma 4.9 can be found in the full paper [20]. The subsequent Lemma offers a helpful formula for $\mathbf{\Pi}_1^{\text{up}}$, the orthogonal projection matrix onto the image of $\boldsymbol{L}_1^{\text{up}}$.

▶ **Lemma 4.10.** *Let $\mathcal{K}$ be a simplicial complex with boundary operator $\partial_2$. For any partition $F \cup C$ of the 2-simplexes of $\mathcal{K}$, the orthogonal projection $\mathbf{\Pi}_1^{up}$ for $\mathcal{K}$ can be decomposed as*

$$\mathbf{\Pi}_1^{up} = \mathbf{\Pi}_{Im(\partial_2[:,F])} + \mathbf{\Pi}_{Ker(\partial_2^\top[F,:])} \partial_2[:,C] (Sc[\boldsymbol{L}_2^{down}]_C)^\dagger \partial_2^\top[C,:] \mathbf{\Pi}_{Ker(\partial_2^\top[F,:])},$$

*where $\boldsymbol{L}_2^{down}$ is the down 2-Laplacian.*

Once more, an $r$-hollowing offers a natural partition of the triangles within $\mathcal{K}$. We assign all the "interior" triangles to $F$ and all the "boundary" triangles to $C$. As such, Nested Dissection can be utilized to compute $\mathbf{\Pi}_{\text{Im}(\partial_2[:,F])}$ and $\mathbf{\Pi}_{\text{Ker}(\partial_2^\top[F,:])}$. The primary technical challenge arises when solving a system in the Schur complement $\text{Sc}[\boldsymbol{L}_2^{\text{down}}]_C$. We precondition it using the boundary $\partial_2^\top[C,:]\partial_2[:,C]$ and apply Preconditioned Conjugate Gradient, which requires a distinct approach to bound the relative condition numbers.

# 5 Proof of Main Theorem 3.1

Given all the four operators in Lemma 4.2, 4.3, 4.4, and 4.9, we prove Theorem 3.1.

**Proof of Theorem 3.1.** Let $\kappa$ be the maximum of $\kappa(\boldsymbol{L}_1^{\text{down}})$ and $\kappa(\boldsymbol{L}_1^{\text{up}})$. Let $\delta > 0$ be a parameter to be determined later. Let $\widetilde{\mathbf{\Pi}}_1^{\text{down}} = \widetilde{\mathbf{\Pi}}_1^{\text{down}}(\delta), \widetilde{\mathbf{\Pi}}_1^{\text{up}} = \widetilde{\mathbf{\Pi}}_1^{\text{up}}(\delta)$ be defined in Lemma 4.2 and 4.9, and let $\boldsymbol{Z}_1^{\text{down}}$ be the operator in Lemma 4.3 with no error and $\boldsymbol{Z}_1^{\text{up}}$ in Lemma 4.4 with error $\delta$. Let

$$\begin{aligned}
&\tilde{\boldsymbol{b}}^{\text{up}} \stackrel{\text{def}}{=} \widetilde{\mathbf{\Pi}}_1^{\text{up}} \boldsymbol{b}, \ \tilde{\boldsymbol{b}}^{\text{down}} \stackrel{\text{def}}{=} \widetilde{\mathbf{\Pi}}_1^{\text{down}} \boldsymbol{b}, \\
&\widetilde{\boldsymbol{x}}^{\text{up}} \stackrel{\text{def}}{=} \boldsymbol{Z}_1^{\text{up}} \tilde{\boldsymbol{b}}^{\text{up}}, \ \widetilde{\boldsymbol{x}}^{\text{down}} \stackrel{\text{def}}{=} \boldsymbol{Z}_1^{\text{down}} \tilde{\boldsymbol{b}}^{\text{down}}, \\
&\tilde{\boldsymbol{x}} \stackrel{\text{def}}{=} \widetilde{\mathbf{\Pi}}_1^{\text{up}} \widetilde{\boldsymbol{x}}^{\text{up}} + \widetilde{\mathbf{\Pi}}_1^{\text{down}} \widetilde{\boldsymbol{x}}^{\text{down}}.
\end{aligned}$$

Then,

$$\begin{aligned}
&\left\| \boldsymbol{L}_1 \tilde{\boldsymbol{x}} - \mathbf{\Pi}_1 \boldsymbol{b} \right\|_2 \\
&\quad \leq \left\| \boldsymbol{L}_1^{\text{up}} \widetilde{\mathbf{\Pi}}_1^{\text{up}} \widetilde{\boldsymbol{x}}^{\text{up}} - \tilde{\boldsymbol{b}}^{\text{up}} \right\|_2 + \left\| \boldsymbol{L}_1^{\text{down}} \widetilde{\mathbf{\Pi}}_1^{\text{down}} \widetilde{\boldsymbol{x}}^{\text{down}} - \tilde{\boldsymbol{b}}^{\text{down}} \right\|_2 + \left\| \tilde{\boldsymbol{b}}^{\text{up}} + \tilde{\boldsymbol{b}}^{\text{down}} - \mathbf{\Pi}_1 \boldsymbol{b} \right\|_2 .
\end{aligned}$$

- For the first term,

$$\left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2 \le \left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\boldsymbol{L}_1^{\text{up}}\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}\right\|_2 + \left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2.$$

  By Lemma 4.9,

$$\begin{aligned}\left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\boldsymbol{L}_1^{\text{up}}\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}\right\|_2 &\le \|\boldsymbol{L}_1^{\text{up}}\|_2\left\|(\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}-\boldsymbol{\Pi}_1^{\text{up}})\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}\right\|_2\\ &\le \delta\|\boldsymbol{L}_1^{\text{up}}\|_2\|\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}\|_2.\end{aligned}$$

  Let $\boldsymbol{y} \stackrel{\text{def}}{=} (\boldsymbol{L}_1^{\text{up}})^{\dagger}\tilde{\boldsymbol{b}}^{\text{up}}$. By Lemma 4.4,

$$\|\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\boldsymbol{y}\|_2 \le \|(\boldsymbol{L}_1^{\text{up}})^{\dagger}\|_2\left\|\boldsymbol{L}_1^{\text{up}}\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2 \le \delta\|(\boldsymbol{L}_1^{\text{up}})^{\dagger}\|_2\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2.$$

  By the triangle inequality,

$$\|\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}\|_2 \le \|\boldsymbol{y}\|_2+\delta\|(\boldsymbol{L}_1^{\text{up}})^{\dagger}\|_2\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2 \le (1+\delta)\|(\boldsymbol{L}_1^{\text{up}})^{\dagger}\|_2\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2.$$

  So, $\left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\boldsymbol{L}_1^{\text{up}}\boldsymbol{\Pi}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}\right\|_2 \le \delta(1+\delta)\kappa\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2$.
  By Lemma 4.4, $\left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2 \le \delta\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2$.
  So, $\left\|\boldsymbol{L}_1^{\text{up}}\widetilde{\boldsymbol{\Pi}}_1^{\text{up}}\widetilde{\boldsymbol{x}}^{\text{up}}-\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2 \le 3\delta\kappa\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2$.
- For the second term, the operator $\boldsymbol{Z}_1^{\text{down}}$ has no error, which means $\boldsymbol{L}_1^{\text{down}}\widetilde{\boldsymbol{x}}^{\text{down}} = \tilde{\boldsymbol{b}}^{\text{down}}$. Then,

$$\begin{aligned}\left\|\boldsymbol{L}_1^{\text{down}}\widetilde{\boldsymbol{\Pi}}_1^{\text{down}}\widetilde{\boldsymbol{x}}^{\text{down}}-\tilde{\boldsymbol{b}}^{\text{down}}\right\|_2 &= \left\|\boldsymbol{L}_1^{\text{down}}\widetilde{\boldsymbol{\Pi}}_1^{\text{down}}\widetilde{\boldsymbol{x}}^{\text{down}}-\boldsymbol{L}_1^{\text{down}}\widetilde{\boldsymbol{x}}^{\text{down}}\right\|_2\\ &\le \delta(1+\delta)\kappa\left\|\tilde{\boldsymbol{b}}^{\text{down}}\right\|_2.\end{aligned}$$

- For the third term,

$$\begin{aligned}\left\|\tilde{\boldsymbol{b}}^{\text{up}}+\tilde{\boldsymbol{b}}^{\text{down}}-\boldsymbol{\Pi}_1\boldsymbol{b}\right\|_2^2 &= \left\|(\widetilde{\boldsymbol{\Pi}}^{\text{up}}-\boldsymbol{\Pi}^{\text{up}})\boldsymbol{b}\right\|_2^2+\left\|(\widetilde{\boldsymbol{\Pi}}^{\text{down}}-\boldsymbol{\Pi}^{\text{down}})\boldsymbol{b}\right\|_2^2\\ &\le \delta^2\left(\|\boldsymbol{\Pi}^{\text{up}}\boldsymbol{b}\|_2^2+\left\|\boldsymbol{\Pi}^{\text{down}}\boldsymbol{b}\right\|_2^2\right) \text{ (by Lemma 4.2, 4.9, Fact 2.5)}\\ &= \delta^2\|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2^2.\end{aligned}$$

  Combining all the above inequalities,

$$\begin{aligned}\|\boldsymbol{L}_1\tilde{\boldsymbol{x}}-\boldsymbol{\Pi}_1\boldsymbol{b}\|_2 &\le 3\delta\kappa\left\|\tilde{\boldsymbol{b}}^{\text{up}}\right\|_2+2\delta\kappa\left\|\tilde{\boldsymbol{b}}^{\text{down}}\right\|_2+\delta\|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2\\ &\le 3\delta\kappa(1+\delta)\|\boldsymbol{\Pi}_1^{\text{up}}\boldsymbol{b}\|_2+2\delta\kappa(1+\delta)\left\|\boldsymbol{\Pi}_1^{\text{down}}\boldsymbol{b}\right\|_2+\delta\|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2\\ &\le 11\delta\kappa\|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2.\end{aligned}$$

  Choosing $\delta \le \frac{\epsilon}{11\kappa}$, we have

$$\|\boldsymbol{L}_1\tilde{\boldsymbol{x}}-\boldsymbol{\Pi}_1\boldsymbol{b}\|_2 \le \epsilon\|\boldsymbol{\Pi}_1\boldsymbol{b}\|_2.$$

◀

## 6 Computing an $r$-Hollowing

In this section, we describe a linear time algorithm (Algorithm 2) that finds an $r$-hollowing of a pure 3-complex $\mathcal{K}$ embedded in $\mathbb{R}^3$ with $n$ stable simplexes that satisfies the additional geometric structures stated at the end of Section 2. We restate them below:

1. The aspect ratio of the convex hull of $\mathcal{K}$ is $O(1)$, and the volume of each tetrahedron in $\mathcal{K}$ is $\Theta(1)$.
2. All but one boundary component has 1-skeleton diameter $O(r^{1/3})$.
3. The total number of exterior simplexes of $\mathcal{K}$ within any $\mathbb{X} \subset \mathbb{R}^3$ of volume $r$ is $O(r^{2/3})$; the total number of exterior simplexes of $\mathcal{K}$ is $O(nr^{-1/3})$.
4. The triangle distance between any two boundary components of $\mathcal{K}$ is greater than 5.

▶ **Lemma 6.1** (Finding an $r$-hollowing). *Let $\mathcal{K}$ be a pure 3-complex embedded in $\mathbb{R}^3$ and composed of $n$ stable simplexes. If $\mathcal{K}$ possesses additional geometric structures 1-4 with parameter $r = o(n)$, then we can find an $r$-hollowing of $\mathcal{K}$ in linear time.*

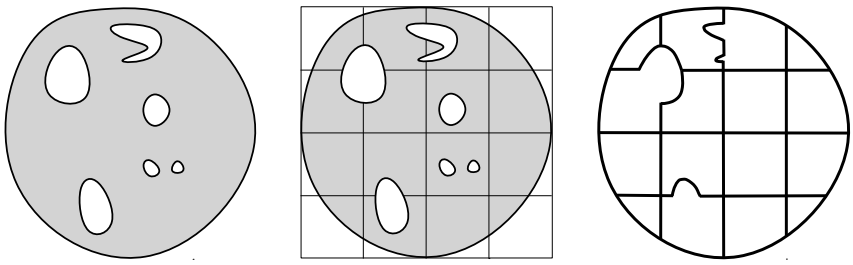
In the rest of the section, we will show Algorithm 2 satisfies Lemma 6.1. Let $\mathbb{K}$ be the convex hull of the underlying topological space of $\mathcal{K}$. Algorithm 2 first finds a *nice bounding box* – a box encompasses $\mathbb{K}$ and its volume and aspect ratio are within constant factors of those of $\mathbb{K}$. Lemma 6.3 provides a linear time algorithm for finding a nice bounding box for $\mathbb{K}$ when the aspect ratio of $\mathbb{K}$ is $O(1)$. Then, Algorithm 2 "cuts" the bounding box into $O(n/r)$ smaller boxes of equal volume using 2-dimensional planes and turns these cutting planes into an $r$-hollowing. Figure 2 illustrates the process of finding an $r$-hollowing.

We need the following lemma from [4] to construct a nice bounding box.

▶ **Lemma 6.2** (Lemma 3.4 of [4]). *Given a set $X$ of points in $\mathbb{R}^3$, we can compute in linear time a bounding box $\mathcal{B}$ with $vol(\mathcal{B}) \leq 6\sqrt{6}vol(\mathcal{B}^*)$, where $vol(\cdot)$ is the volume and $\mathcal{B}^*$ is a bounding box of $X$ with the minimum volume.*

▶ **Corollary 6.3** (Nice bounding box). *Let $\mathcal{K}$ be a 3-complex embedded in $\mathbb{R}^3$ whose underlying topological space has aspect ratio $O(1)$. We can compute a nice bounding box of the convex hull of $\mathcal{K}$ in linear time.*

The proof of Corollary 6.3 can be found in the full version [20].



(a) (b) (c)

**Figure 2** (a) An 2-dimensional illustration of a 3-complex $\mathcal{K}$ with several holes inside; (b) A nice bounding box of $\mathcal{K}$ with $\lfloor n^{1/3}r^{-1/3} \rfloor$ evenly-spaced 2-dimensional planes; (c) An $r$-hollowing $\mathcal{T}$ generated by Algorithm 2 consisting of simplexes that are "close" to the two-dimensional planes and parts of the boundaries of the intersecting holes inside with the planes.

**Proof of Lemma 6.1.** We can check that Algorithm 2 has linear runtime. In the rest of the proof, we will show $\mathcal{T}$ returned by Algorithm 2 is an $r$-hollowing of $\mathcal{K}$.

By Assumption 1, 2 and 3, the volume of the convex hull of $\mathcal{K}$, denoted by $\mathrm{CH}(\mathcal{K})$, is $\Theta(n)$; the maximum volume is attained when $\mathcal{K}$ has $\Theta(n/r)$ boundary components and each corresponds to a "hole" of volume $\Theta(r)$. By Lemma 6.3, we have $\mathrm{vol}(\mathcal{B}) = \Theta(\mathrm{vol}(\mathrm{CH}(\mathcal{K}))) = \Theta(n)$. In Algorithm 2, the 2-dimensional planes divide the box $\mathcal{B}$ into $O(n/r)$ smaller boxes

**Algorithm 2** HOLLOWING$(\mathcal{K}, r)$.

**Input:** A pure 3-complex $\mathcal{K}$ embedded in $\mathbb{R}^3$ with $n$ stable simplexes satisfying assumption 1-4 with a known parameter $r = o(n)$

**Output:** An $r$-hollowing $\mathcal{T}$

Find a nice bounding box $\mathcal{B}$ for $\mathcal{K}$ by Corollary 6.3.

For each pair of parallel faces of $\mathcal{B}$, find $\lfloor n^{1/3} r^{-1/3} \rfloor$ evenly-spaced 2-dimensional planes parallel to the face which divide $\mathcal{B}$ into equal-volume pieces. We can slightly perturb the planes so that no plane intersects with any vertex of $\mathcal{K}$ (see Figure 2(b)).

$\mathcal{T} \leftarrow$ all the tetrahedrons on the boundary of $\mathcal{K}$ that form a spherical shell.

**for** *each 2-dimensional plane* $P$ **do**

  $\mathcal{Q} \leftarrow$ all the tetrahedrons of $\mathcal{K}$ that intersect $P$.

  **if** $\mathcal{Q}$ *is not connected (i.e.,* $P$ *intersects some holes inside)* **then**

    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \bigcup_{\mathcal{H} \in \text{intersected holes inside}}$ all the tetrahedrons on the boundary of $\mathcal{H}$, which are on one side of $P$ and form half of a spherical shell (see Figure 2(c)).

  **end**

  $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{Q}$.

**end**

Expand $\mathcal{T}$ such that its width reaches 5.

**return** $\mathcal{T}$

each of volume $O(r)$ and surface area $O(r^{2/3})$. By our construction of $\mathcal{T}$, each smaller box corresponds to a region; thus, there are $O(n/r)$ regions. By Assumption 3, each region of $\mathcal{T}$ has $O(r)$ simplexes and $O(r^{2/3})$ boundary simplexes. Moreover, the boundary of each region triangulates a spherical shell in $\mathbb{R}^3$ by construction. Additionally, the diameter of the underlying topological space of each region is upper bounded by the triangle diameter of the small box plus $\Theta(1)$ times the 1-skeleton diameter of boundary components. By Assumption 2, each region has diameter $O(r^{1/3})$.

To conclude, $\mathcal{T}$ satisfies all the conditions in Definition 2.9 and is an $r$-hollowing of $\mathcal{K}$. ◀

## References


1 Owe Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25(1):165–187, 1985.

2 Mitali Bafna and Nikhil Vyas. Optimal fine-grained hardness of approximation of linear equations. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

3 Sergio Barbarossa and Stefania Sardellitti. Topological signal processing over simplicial complexes. *IEEE Transactions on Signal Processing*, 68:2992–3007, 2020.

4 Gill Barequet and Sariel Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38(1):91–109, 2001.

5 Mitchell Black, William Maxwell, Amir Nayyeri, and Eli Winkelman. Computational topology in a collapsing universe: Laplacians, homology, cohomology. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 226–251. SIAM, 2022.

6 Mitchell Black and Amir Nayyeri. Hodge decomposition and general laplacian solvers for embedded simplicial complexes. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

**7** Erik G Boman, Bruce Hendrickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis*, 46(6):3264–3284, 2008.

**8** Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

**9** Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

**10** Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022.

**11** David RJ Chillingworth. Collapsing three-dimensional convex polyhedra. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 63, pages 353–357. Cambridge University Press, 1967.

**12** David RJ Chillingworth. Collapsing three-dimensional convex polyhedra: correction. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 88, pages 307–310. Cambridge University Press, 1980.

**13** Michael B Cohen, Brittany Terese Fasy, Gary L Miller, Amir Nayyeri, Richard Peng, and Noel Walkington. Solving 1-laplacians in nearly linear time: Collapsing and expanding a topological ball. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 204–216. SIAM, 2014.

**14** Michael B Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B Rao, and Aaron Sidford. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 898–909. IEEE, 2018.

**15** Michael B Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 410–419, 2017.

**16** Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly $m \log^{1/2} n$ time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 343–352. ACM, 2014.

**17** Samuel I Daitch and Daniel A Spielman. Support-graph preconditioners for 2-dimensional trusses. *arXiv preprint cs/0703119*, 2007.

**18** Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 451–460, 2008.

**19** Ming Ding, Rasmus Kyng, Maximilian Probst Gutenberg, and Peng Zhang. Hardness results for laplacians of simplicial complexes via sparse-linear equation complete gadgets. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

**20** Ming Ding and Peng Zhang. Weighted 1-laplacian solvers for well-shaped simplicial complexes. *arXiv preprint arXiv:2302.06499*, 2023.

**21** Beno Eckmann. Harmonische Funktionen und Randwertaufgaben in einem Komplex. *Commentarii Mathematici Helvetici*, 17(1):240–255, December 1944. doi:10.1007/BF02566245.

**22** Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Soc., 2010.

23 Herbert Edelsbrunner and Salman Parsa. On the computational complexity of betti numbers: reductions from matrix rank. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 152–160. SIAM, 2014.

24 Joel Friedman. Computing betti numbers via combinatorial laplacians. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, pages 386–391, 1996.

25 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

26 Robert Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

27 Arun Jambulapati and Aaron Sidford. Ultrasparse ultrasparsifiers and faster laplacian system solvers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 540–559. SIAM, 2021.

28 Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. Statistical ranking and combinatorial hodge theory. *Mathematical Programming*, 127(1):203–244, 2011.

29 Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920, 2013.

30 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 235–244, Washington, DC, USA, 2010. IEEE Computer Society. doi:10.1109/FOCS.2010.29.

31 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/FOCS.2011.85.

32 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 842–850, 2016.

33 Rasmus Kyng, Richard Peng, Robert Schwieterman, and Peng Zhang. Incomplete nested dissection. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 404–417, 2018.

34 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 573–582. IEEE, 2016.

35 Hyekyoung Lee, Moo K Chung, Hongyoon Choi, Hyejin Kang, Seunggyun Ha, Yu Kyeong Kim, and Dong Soo Lee. Harmonic holes as the submodules of brain network and network dissimilarity. In *Computational Topology in Image Context: 7th International Workshop, CTIC 2019, Málaga, Spain, January 24-25, 2019, Proceedings 7*, pages 110–122. Springer, 2019.

36 Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *2013 ieee 54th annual symposium on foundations of computer science*, pages 147–156. IEEE, 2013.

37 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

38 Lek-Heng Lim. Hodge laplacians on graphs. *Siam Review*, 62(3):685–715, 2020.

39 Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979.

40 Wenye Ma, Jean-Michel Morel, Stanley Osher, and Aichi Chien. An l 1-based variational model for retinex theory and its application to medical images. In *CVPR 2011*, pages 153–160. IEEE, 2011.

**41** Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013.

**42** Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

**43** Gary L Miller and William Thurston. Separators in two and three dimensions. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 300–309, 1990.

**44** Zipei Nie. Matrix anti-concentration inequalities with applications. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 568–581, 2022.

**45** Braxton Osting, Jérôme Darbon, and Stanley Osher. Statistical ranking using the l1-norm on graphs. *AIMS Journal on Inverse Problems and Imaging*, 7(3):907–926, 2013.

**46** Richard Peng and Daniel A Spielman. An efficient parallel solver for sdd linear systems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 333–342. ACM, 2014.

**47** Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 504–521. SIAM, 2021.

**48** Gil Shklarski and Sivan Toledo. Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures. *SIAM Journal on Matrix Analysis and Applications*, 30(1):7–40, 2008.

**49** Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 563–568, 2008.

**50** Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.

**51** Martin Tancer. Recognition of collapsible complexes is np-complete. *Discrete & Computational Geometry*, 55(1):21–38, 2016.

**52** Shang-Hua Teng. The laplacian paradigm: Emerging algorithms for massive graphs. In *Theory and Applications of Models of Computation: 7th Annual Conference, TAMC 2010, Prague, Czech Republic, June 7-11, 2010. Proceedings 7*, pages 2–14. Springer, 2010.

**53** Yiying Tong, Santiago Lombeyda, Anil N Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM transactions on graphics (TOG)*, 22(3):445–452, 2003.

**54** Ke Ye and Lek-Heng Lim. Cohomology of cryo-electron microscopy. *SIAM Journal on Applied Algebra and Geometry*, 1(1):507–535, 2017.

# Optimal Energetic Paths for Electric Cars

**Dani Dorfman**
Tel Aviv University, Israel

**Haim Kaplan**
Tel Aviv University, Israel

**Robert E. Tarjan**
Princeton University, NJ, USA

**Uri Zwick**
Tel Aviv University, Israel

## Abstract

A weighted directed graph $G = (V, A, c)$, where $A \subseteq V \times V$ and $c : A \to \mathbb{R}$, naturally describes a road network in which an electric car, or vehicle (EV), can roam. An arc $uv \in A$ models a road segment connecting the two vertices (junctions) $u$ and $v$. The cost $c(uv)$ of the arc $uv$ is the amount of *energy* the car needs to travel from $u$ to $v$. This amount can be positive, zero or negative. We consider both the more realistic scenario where there are no negative cycles in the graph, as well as the more challenging scenario, which can also be motivated, where negative cycles may be present.

The electric car has a battery that can store up to $B$ units of energy. The car can traverse an arc $uv \in A$ only if it is at $u$ and the charge $b$ in its battery satisfies $b \geq c(uv)$. If the car traverses the arc $uv$ then it reaches $v$ with a charge of $\min\{b - c(uv), B\}$ in its battery. Arcs with a positive cost deplete the battery while arcs with negative costs may charge the battery, but not above its capacity of $B$. If the car is at a vertex $u$ and cannot traverse any outgoing arcs of $u$, then it is stuck and cannot continue traveling.

We consider the following natural problem: Given two vertices $s, t \in V$, can the car travel from $s$ to $t$, starting at $s$ with an initial charge $b$, where $0 \leq b \leq B$? If so, what is the maximum charge with which the car can reach $t$? Equivalently, what is the smallest *depletion* $\delta_{B,b}(s, t)$ such that the car can reach $t$ with a charge of $b - \delta_{B,b}(s, t)$ in its battery, and which path should the car follow to achieve this? We also refer to $\delta_{B,b}(s, t)$ as the *energetic cost* of traveling from $s$ to $t$. We let $\delta_{B,b}(s, t) = \infty$ if the car cannot travel from $s$ to $t$ starting with an initial charge of $b$. The problem of computing energetic costs is a strict generalization of the standard shortest paths problem.

When there are no negative cycles, the single-source version of the problem can be solved using simple adaptations of the classical Bellman-Ford and Dijkstra algorithms. More involved algorithms are required when the graph may contain negative cycles.

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

**Keywords and phrases** Electric cars, Optimal Paths, Battery depletion

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.42

**Funding** *Dani Dorfman*: Supported by ISF grant 2854/20.
*Haim Kaplan*: Supported by ISF grant 1595/19 and the Blavatnik Family Foundation.
*Robert E. Tarjan*: Partially supported by a gift from Microsoft.
*Uri Zwick*: Supported by ISF grant 2854/20.

**Acknowledgements** We would like to thank the anonymous reviewers for their comments, for pointing out reference [14], and for pointing out the relation to the 1-VASS problem.

## 1 Introduction

A weighted directed graph $G = (V, A, c)$, where $A \subseteq V \times V$ and $c : A \to \mathbb{R}$, naturally describes a road network in which an electric car can roam. An arc $uv \in A$ models a road segment connecting the two vertices (junctions) $u$ and $v$. The cost $c(uv)$ of the arc $uv$ is the





amount of *energy* the electric car needs to travel from $u$ to $v$. This amount can be positive, e.g., if the road segment is uphill; zero; or negative, e.g., if the road segment is downhill. We consider both the more realistic scenario where there are no negative cycles in the graph, as well as the more challenging scenario, which can also be motivated, where the graph may contain negative cycles. (A cycle is negative if the sum of its arc costs is negative.)

An electric car is equipped with a battery that can store up to $B$ units of energy, where $B > 0$ is a parameter. We assume that the electric car cannot be charged along the way and has to rely on the initial charge available in its battery. If the car is currently at vertex $u$ with charge $b$ in its battery, where $0 \leq b \leq B$, then it can traverse an arc $uv \in A$ if and only if $c(uv) \leq b$. If this condition holds, and the car traverses the arc, then it reaches $v$ with a charge of $\min\{b - c(uv), B\}$. In particular, the charge in the battery cannot be negative and cannot exceed $B$. The car can traverse $uv$ if $b - c(uv) > B$ (which can hold only if $c(uv) < 0$), but the battery will not charge beyond its capacity of $B$. We may assume that $c(uv) \in [-B, B]$ for every $uv \in A$, as arcs with $c(uv) > B$ can never be used, and thus can be removed, and costs $c(uv) < -B$ can be changed to $c(uv) = -B$.

We consider the following natural problem: Given two vertices $s, t \in V$, can the car travel from $s$ to $t$, starting at $s$ with an initial charge $b$, where $0 \leq b \leq B$? If so, what is the *maximum final charge* $\alpha_{B,b}(s,t)$ with which the car can reach $t$? Equivalently, what is the *minimum depletion* $\delta_{B,b}(s,t)$ such that the car can reach $t$ with a charge of $b - \delta_{B,b}(s,t)$ in its battery, and which path should the car follow to achieve this? (If $b < B$ then the minimum depletion $\delta_{B,b}(s,t)$ may be negative.) We also refer to $\delta_{B,b}(s,t)$ as the *energetic cost* of traveling from $s$ to $t$. Note that $\delta_{B,b}(s,t) = b - \alpha_{B,b}(s,t)$. We let $\alpha_{B,b}(s,t) = -\infty$ and $\delta_{B,b}(s,t) = \infty$ if the car cannot travel from $s$ to $t$ starting with an initial charge of $b$.

We also consider the related problem of finding the *minimum initial change* at $s$, if any, that will allow the car to travel to $t$, ending with a charge of at least $b$ in the battery. We denote this quantity by $\beta_{B,b}(s,t)$. We show that minimum initial charges can be computed by computing maximum final charges, or minimum energetic costs, on the *reversed* graph.

If all arc costs are non-negative, then $\delta_{B,b}(s,t) = \delta(s,t)$, if $\delta(s,t) \leq b \leq B$, where $\delta(s,t)$ is the standard distance, i.e., the length of a shortest path, from $s$ to $t$ in the graph $G$, where $c(uv)$ is the length of $uv$. Otherwise, $\delta_{B,b}(s,t) = \infty$. If there are negative arc costs but no negative cycles, $\delta_{B,b}(s,t) = \delta(s,t)$ if and only if there exists a shortest path $P$ from $s$ to $t$ such that the length of every prefix of $P$ is in the interval $[b - B, b]$. In general, energetic costs may be larger than distances, since the charge in the battery is constrained to remain in the interval $[0, B]$, i.e., it is not allowed to go negative and it is capped at $B$. (For example, the electric car may not be able to traverse a mountain pass and may need to take a detour.) It is always true that $\delta_{B,b}(s,t) \geq \delta(s,t)$.

The problem of computing minimum energetic costs is thus a strict generalization of the standard shortest paths problem, even if there are no negative cycles in the graph. Interestingly, the energetic costs $\delta_{B,b}(s,t)$ are well-defined even if there are negative cycles in the graph. The problem is then an interesting variant of one-player *energy games* with a reachability objective. It is also related to the 1-VASS (Vector Addition Systems with States) problem. (See references Section 1.2.) The presence of negative cycles poses interesting algorithmic challenges. The corresponding minimum energetic paths are still finite, but are not necessarily simple. A minimum energy path from $s$ to $t$ may need to go through a sequence of negative cycles, using each one of them to gain sufficient energy to reach the next negative cycle, and eventually the target $t$.

When there are no negative cycles, the single-source version of the energetic cost problem can be solved using simple, but subtle, adaptations of the classical Bellman-Ford [4, 15] and Dijkstra [11] algorithms. Similar algorithms, some less efficient, are explicit or implicit

in previous papers (see Section 1.2). We present simpler, self-contained versions of these algorithms with simpler correctness proofs. We also present efficient algorithms for finding minimum energetic paths in the presence of negative cycles.

Unlike the standard shortest paths problem, the single-target version of the minimum energetic paths problem is *not* equivalent to the single-source version. In particular, one cannot solve the single-target problem by running a single-source algorithm backward. Although there is always a tree of minimum energetic paths from a source vertex $s$ to all other vertices reachable from it, there are simple examples in which there is no tree of minimum energetic paths to a target vertex $t$ from all vertices that can reach it.

## 1.1 Our results

We show that the single-source version of the minimum energetic paths problem with negative arc costs but no negative cycles can be solved in $O(mn)$ time using a simple adaptation of the Bellman-Ford [4, 15] algorithm, where $m = |A|$ and $n = |V|$. Furthermore, if a *valid potential function* $p : V \to \mathbb{R}$ is given, i.e., a function for which $c(uv) + p(u) - p(v) \geq 0$ for every $uv \in A$, then the single-source version can be solved in $O(m + n \log n)$ time using an adaptation of Dijkstra's [11] algorithm equivalent to the $A^*$ search heuristic (see, e.g., Hart et al. [21]). Since a valid potential function can be found in $O(mn)$ time using the standard Bellman-Ford algorithm, the all-pairs version of the minimum energetic paths problem can be solved $O(mn + n^2 \log n)$ time.

The $O(mn)$ bound matches the time bound of the standard Bellman-Ford algorithm, which is still the fastest known algorithm for the single-source shortest paths problem in a directed graph with general (real) arc costs, and no negative cycles. The $O(mn + n^2 \log n)$ bound almost matches the best time bound of $O(mn + n^2 \log \log n)$ obtained by Pettie [31] for the standard APSP problem with general arc costs. (For a survey of other related results, see Zwick [38].)

Much faster algorithms are known for the standard single-source shortest paths problem when arc costs are integral. Bringmann et al. [8], improving a breakthrough result of Bernstein et al. [5], obtained an $O(m \log^2 n \log(nW) \log \log n)$-time algorithm when arc costs are integers that are at least $-W$, where $W \geq 1$. By using these algorithms to find a valid potential function, and then using the energetic version of Dijkstra's algorithm, we get the same improved time bound for the single-source version of the minimum energetic paths problem with negative arc costs but no negative cycles.

We also present a more involved $O(mn + n^2 \log n)$-time algorithm for solving the single-source minimum energetic cost problem in the presence of negative cycles. This gives, of course, an $O(mn^2 + n^3 \log n)$-time algorithm for the all-pairs and single-target versions of the problem.

When the capacity $B$ of the battery is sufficiently large, we show that the all-pairs version of the minimum energetic cost problem can be solved in $O(mn + n^2 \log n)$ time.

## 1.2 Related results

Various adaptations of the Bellman-Ford and Dijkstra algorithms for problems similar or equivalent to the minimum energetic paths problem defined here, when there are no negative cycles in the graph, were given by several authors. Eisner et al. [14], improving upon Artmeier et al. [2], and Brim and Chalupka [6] give versions of these algorithms with the same running times as ours, but using a slightly different approach. Baum et al. [3] give a version of Dijkstra's algorithm but with a much slower running time. They also show that the maximum charge with which $t$ can be reached when starting at $s$ with charge $b$ is a piece-wise linear function of $b$ with at most $O(n)$ breakpoints.

Brim and Chalupka [6] consider the related problem of solving one-player *energy games* and the more complicated problem of solving two-player *energy games*. (For more on energy games, see also Brim et al. [7] and Dorfman et al. [12].) The problem of finding minimum energy paths in the presence of negative cycles may be seen as a variant of one-player energy games with a reachability objective. The minimum energetic paths problem is also related to the 1-VASS problem. (See, e.g., Almagor et al. [1] and Künnemann et al. [26].)

Khuller et al. [23] consider a related problem in which the battery (or the fuel tank) can be recharged at intermediate vertices, with a possibly different price per unit of charge at each intermediate vertex. All arc costs are non-negative. They give various algorithms for computing a cheapest path from $s$ to $t$. Among these algorithms is a $O(n^2\Delta \log n)$-time algorithm for the single-target version, where $\Delta$ is a bound on the number of rechargings allowed, and an $O(n^3)$-time algorithm for the single-target version when the number of rechargings is unbounded.

Several authors, including Lehmann [27], Tarjan [33] and Mohri [30] considered generalized versions of the shortest paths problem defined by *semirings*. If $(R, \oplus, \otimes)$ is a semiring and $P$ is an $s$-$t$ path whose arcs have costs $c_i$, the cost of $P$ is defined to be $c(P) = \otimes_{i=1}^{k} c_i$. The goal is to find $\oplus_P c(P)$, where $P$ ranges over all $s$-$t$ paths, assuming this quantity is well defined. The standard shortest paths problem corresponds to the *tropical* semiring $(\mathbb{R}, \min, +)$. All these results assume, as part of the definition of a semiring, that $\otimes$ is associative. Thus, as we shall see, none of these results apply to our problem, as our operation $\otimes$ is not associative.

Generalized versions of Dijkstra's algorithm were obtained by various authors, most notably by Knuth [25]. These generalizations are of a different nature and are apparently not related to the version given here.

Other non-standard versions of the shortest paths problem were also considered. Perhaps the most famous one is the *bottleneck* shortest paths problem. See, e.g., [17, 9] for the single-pair version, and [36, 13] for the all-pairs version. Vassilevska [35] considered an interesting non-standard *non-decreasing* version of the shortest paths problem related to reading train schedules. Finally, Madani et al. [29] considered the *discounted* shortest paths problem. All these problems are quite different from the problem considered here.

## 2 Minimum energetic paths

To simplify the presentation, we concentrate on the computation of $\delta_B(s,t) = \delta_{B,B}(s,t)$, i.e., the energetic cost of traveling from $s$ to $t$ when starting with a fully charged battery of capacity $B$. Computing $\delta_{B,b}(s,t)$, for an arbitrary $0 \le b \le B$, can easily be reduced to computing $\delta_B(s,t)$: Add a new vertex $s'$ and an arc $s's$ of cost $c(s's) = B - b$. Then, it is easy to see that $\delta_{B,b}(s,t) = \delta_B(s',t) - (B-b)$. A similar idea can be incorporated directly into the algorithms that we describe. We begin by defining the energetic cost of a path.

▶ **Definition 2.1** (Energetic cost of a path). *A path $P = u_0u_1 \dots u_k$ is* traversable *if it can be traversed when starting from $u_0$ with a fully charged battery. If $P$ is traversable, the final charge in the battery when reaching $u_k$ is $B - d_B(P)$, where $d_B(P)$ is defined to be the* depletion*, or the* energetic cost *of the path. Note that $0 \le d_B(P) \le B$. If the path is not traversable, we let $d_B(P) = \infty$.*

To obtain a simple formula for $d_B(P)$ we define the following operations:

$$x \oplus_B y \;=\; [x+y]_0^B \quad , \quad [z]_0^B \;=\; \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } 0 \le z \le B \\ \infty & \text{otherwise} \end{cases}$$

We assume that $x+\infty = \infty+x = \infty$ for every $x \in [-B, B] \cup \{\infty\}$. For brevity, we sometimes write $x \oplus y$ instead of $x \oplus_B y$ when $B$ is understood from the context. [1] Note that for every $x, y \in \mathbb{R}$ and $B > 0$ we have $x+y \le x \oplus_B y$. It is important to note that $\oplus_B$ is *not* associative. (For example, $B \oplus (B \oplus -B) = B$ while $(B \oplus B) \oplus -B = \infty$, and $(-1 \oplus -2) \oplus 2 = 2$ while $-1 \oplus (-2 \oplus 2) = 0$, assuming $B \ge 2$.)

▶ **Lemma 2.2.** *Let $P = u_0 u_1 \dots u_k$ be a directed path, let $P' = u_0 \dots u_{k-1}$, and let $c_i = c(u_{i-1}u_i)$, for $i = 0, 1, \dots, k$. Let $B$ be the capacity and initial charge of battery at $u_0$. If $k = 0$ then $d_B(P) = 0$. If $k > 0$ then*

$$d_B(P) \;=\; d_B(P') \oplus c_k \;=\; ((\cdots((0 \oplus c_1) \oplus c_2) \oplus \cdots) \oplus c_{k-1}) \oplus c_k \;.$$

**Proof.** Let $b_i$ be the charge of the battery at $u_i$ and let $d_i = B - b_i$ be the depletion of the battery at $u_i$, for $i = 0, 1, \dots, k$. Clearly $d_0 = 0$. It is easy to prove by induction that $d_i = d_{i-1} \oplus_B c_i$. The lemma follows. ◀

As mentioned, the operation $\oplus_B$ is *not* associative. Thus, in general, $d_B(P) \ne 0 \oplus (c_1 \oplus (\cdots \oplus (c_{k-2} \oplus (c_{k-1} \oplus c_k)) \cdots))$. In Section 7 we show, however, that $c_1 \oplus (c_2 \oplus (\cdots \oplus (c_{k-1} \oplus (c_k \oplus 0)) \cdots))$ also has an interesting meaning.

▶ **Definition 2.3** (Energetic costs, minimum energetic paths)**.** *The* energetic cost $\delta_B(s,t)$ *of traveling from $s$ to $t$, starting from $s$ with a fully charged battery of capacity $B$, is defined as*

$$\delta_B(s,t) \;=\; \min\{\, d_B(P) \mid P \text{ is an } s\text{-}t \text{ path in } G \,\} \;.$$

*If $\delta_B(s,t) < \infty$ and $P$ is an $s$-$t$ path satisfying $\delta_B(s,t) = d_B(P)$, then $P$ is said to be a* minimum energetic path *from $s$ to $t$.*

If there are no negative cycles in the graph, then for every path $P$ from $s$ to $t$ there is a simple path $P'$ such that $d_B(P') \le d_B(P)$. (It is in fact enough to require that there are no traversable negative cycles in the graph.) Thus, the minimum in the definition above can be taken over simple paths only. The definition of $\delta_B(s,t)$ is also meaningful in the presence of negative cycles, though minimum energetic paths are not necessarily simple.

It is not difficult to see, and it will also follow from the correctness of the algorithms that we present in the next sections, that for every source vertex $s \in V$ there is always a tree of minimum energetic paths to all other vertices that can be reached from it. The simple example given in Figure 1 shows that a tree of minimum energetic paths to a given target vertex $t$ does not always exist.

To deal with negative cycles, we need the following definition.

▶ **Definition 2.4** (Entry-exit pairs)**.** *Let $C$ be a negative cycle in $G = (V, A, c)$ and let $B$ be the maximum capacity of the battery. A pair of vertices $(x, y)$ on $C$ is an* entry-exit *pair of $C$ if the car can start at $x$ with an empty battery and eventually reach $y$, possibly after going several times around the cycle, with a full battery, i.e., with a charge of $B$.*

The following lemma, similar to the *gasoline puzzle* of Lovász [28, p. 31] (see also Klarner [24, p. 283] and Winkler [37, p. 2]), says that every negative cycle has an entry-exit pair.

▶ **Lemma 2.5.** *Any negative cycle $C$ in $G = (V, A, c)$ contains at least one entry-exit pair. Such a pair can be found in $O(|C|)$ time.*

[1] Note that $\oplus$ is not related to the semiring framework mentioned in Section 1.2.

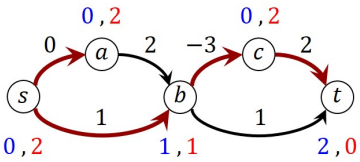



**Figure 1** A simple but illustrative example. Assume $B \geq 3$. The two numbers next to each vertex $u$ are $\delta_B(s,u)$ and $\delta_B(u,t)$. The bold arcs constitute a tree of minimum energetic paths from the source vertex $s$ to all other vertices. Another such tree can be obtained by replacing the arc $ct$ by $bt$. On the other hand, the only minimum path from $a$ to $t$ is $abct$, while the only minimum path from $b$ to $t$ is the arc $bt$. Thus, there is no tree of minimum paths to $t$ from all other vertices.

**Proof.** We show first that every negative cycle has an entry $x$, i.e., a vertex $x$ from which the cycle can be traversed, starting with an empty battery, when the capacity of the battery is ignored. We next show that the entry $x$ has a corresponding exit $y$ when the capacity of the battery is not ignored.

Let $C = v_0 v_1 \cdots v_{\ell-1} v_0$ be a negative cycle in $G$ of length $\ell$. Let $c_j = c(v_j v_{j+1})$, for $j = 0, 1, \ldots, \ell-1$. (We let $v_\ell = v_0$.) For $j \geq \ell$, let $c_j = c_{j \bmod \ell}$. Let $s_i = \sum_{j=0}^{i-1} c_j$ be the prefix sums of the costs around the cycle, starting from $v_0$. (We allow $i > \ell$ by wrapping around the cycle.) Note that $s_\ell = \sum_{j=0}^{\ell-1} c_j < 0$ as the cycle is negative. This also implies that $s_{\ell+i} < s_i$ for every $i$. Vertex $v_0$ is an entry if and only if $s_1, \ldots, s_{\ell-1} \leq 0$. If $v_0$ is not an entry, let $k$ be the index for which $s_k$ is maximized. Note that $0 \leq k < \ell$. We claim that $v_k$ is an entry. Let $s'_i = \sum_{j=0}^{i-1} c_{k+j}$ be the prefix sums starting from $v_k$. Then $s'_i = s_{k+i} - s_k \leq 0$, by the definition of $k$, for every $i$.

Assume, without loss of generality, that $v_0$ is an entry on $C$. Let $s_i = \sum_{j=0}^{i-1} c_j$ as above and let $i = \min\{j \mid s_j \leq -B\} (\mathrm{mod}\ \ell)$. Then $(v_0, v_i)$ is an entry-exit pair. It is not difficult to find $i$ in $O(\ell)$ time. ◀

The following lemma characterizes the structure of minimum energetic paths when the graph may contain negative cycles. It is not difficult to give a direct proof of the lemma. Its correctness also follows from the correctness of the algorithms that we present.

▶ **Lemma 2.6.** *If there is an $s$-$t$ path $P$ with $d_B(P) \leq B$, then there is an $s$-$t$ path $P'$ such that $d_B(P') \leq d_B(P)$ and such that $P'$ has the following form (see Figure 2): either $P'$ is simple, or there is a sequence $C_1, C_2, \ldots, C_k$ of simple negative cycles, where $k < n$, with entry-exit pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$ on them, such that $P'$ is composed of a simple path from $s$ to $x_1$, followed by sufficiently many traversals of $C_1$ that end in $y_1$ with a full battery, followed by a simple path from $y_1$ to $x_2$, followed by sufficiently many traversals of $C_2$ that end in $y_2$ with a full battery, and so on, and finally a simple path from $y_k$ to $t$. Furthermore, all entries $x_1, x_2, \ldots, x_k$ are distinct, and all exits $y_1, y_2, \ldots, y_k$ are distinct.*

## 3 An energetic version of the Bellman-Ford algorithm

Recall that for any $x$ and $y$, $x \oplus_B y \geq x + y$. This implies that in a graph without negative cycles, if there is a traversable path from $s$ to $t$, there is such a path that is simple and hence contains at most $n-1$ arcs. This means that if there are no negative cycles, we can solve the single-source minimum energetic paths problem using the Ford-Bellman [4, 15] shortest path algorithm: We merely replace $+$ by $\oplus_B$.

We base our description of the algorithm on that in [34], which uses a queue as suggested by Gilsinn and Witzgall [18].

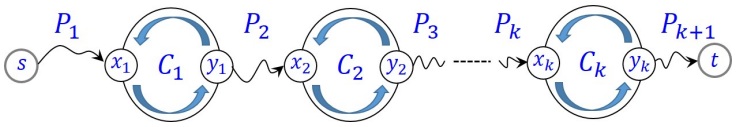



**Figure 2** Generic structure of minimum energetic paths in the presence of negative cycles. If $\delta_B(s,t) \leq B$, then there is a minimum energetic path from $s$ to $t$ of the form shown, where $C_1, \ldots, C_k$ are simple negative cycles and $(x_i, y_i)$ is an entry-exit pair on $C_i$, for $i = 1, 2, \ldots, k$. All entries $x_1, x_2, \ldots, x_k$ are distinct and all exits $y_1, y_2, \ldots, y_k$ are distinct. The paths $P_1, P_2, \ldots, P_{k+1}$ are simple but necessarily disjoint from the cycles $C_1, C_2, \ldots, C_k$.

The algorithm maintains a tentative energetic cost $d(v)$ for each vertex $v$, equal to the minimum of the energetic costs of paths from $s$ to $v$ found so far. Initially $d(s) = 0$ and $d(v) = \infty$ for $v \neq s$, where $s$ is the source. It also maintains a queue $Q$, initially containing $s$. The algorithm repeats the following step until $Q$ is empty:

> Scan a vertex: Delete the front vertex $u$ on $Q$. For each arc $uv$, if $\delta(u) \oplus_B c(uv) < \delta(v)$, *relax* $uv$: Set $\delta(v) \leftarrow \delta(u) \oplus_B c(uv)$, set $\pi(v) \leftarrow u$, and add $v$ to the back of $Q$ if it is not on $Q$.

Pseudocode of the algorithm, which we call *e-BF*, is given on the left of Figure 3. The correctness proof and analysis of the standard Bellman-Ford algorithm in the absence of negative cycles (see e.g., Tarjan [34]) translates directly to this version.

▶ **Theorem 3.1.** *If $G = (V, A, c)$ has no traversable negative cycles then e-BF finds minimum energetic paths from $s$ to all vertices in $O(mn)$ time.*

**Proof.** We define *passes* over the queue. Pass 0 is the first scan step of $s$. Given that pass $k$ is defined, pass $k+1$ is the sequence of scan steps of vertices added to $Q$ during pass $k$. A straightforward induction on $k$ shows that for each vertex $v$ that has a minimum-energy path of at most $k$ arcs, $d(v)$ is the energetic cost of such a path after $k$ passes. It follows that the energetic costs are correctly computed. The $\pi$ values computed describe a tree of minimum-energy paths from $s$ to all vertices reachable from $s$ using a fully charged battery of capacity $B$. Since each pass takes $O(m)$ time, the total time of the algorithm is $O(mn)$. ◀

In addition to the non-existence of negative cycles, the only thing required for correctness of the algorithm is that $\oplus_B$ is non-decreasing in its second argument: If $y \leq z$, $x \oplus y \leq x \oplus z$.

As in the standard version of the Bellman-Ford algorithm, one can add *subtree disassembly* [32, 10], which does not improve the worst-case time bound but is likely to speed up the algorithm in practice. It is also easy to modify the algorithm so that it finds a traversable negative cycle that can be reached from $s$, if one exists.

The correctness of *e-BF* implies the following corollary, which we use to prove the correctness of the energetic variant of Dijkstra's algorithm:

▶ **Corollary 3.2.** *If each $d(v) < \infty$ corresponds to the energetic cost of some path from $s$ to $v$, and $d(v) \leq d(u) \oplus_B c(uv)$ for every $uv \in A$, then $d(v) = \delta_B(s, v)$, for every $v \in V$.*

## 4 An energetic version of Dijkstra's algorithm

If all arc costs are non-negative, Dijkstra's algorithm [11] with $+$ replaced by $\oplus_B$ will solve the single-source problem. This algorithm replaces the queue $Q$ in the Bellman-Ford algorithm by a heap $H$. The key of a vertex $v$ in the heap is $d(v)$. Each scan step deletes a vertex of minimum key from the heap. When a relaxation decreases the key of a vertex in the heap,

```
e-BF(G = (V, A, c), B, s):
    for u ∈ V do
        d(u) ← ∞
        π(u) ← null
    d(s) ← 0
    Q ← Queue()
    Q.insert-last(s)
    while Q ≠ ∅ :
        u ← Q.DeleteFirst()
        for uv ∈ A do
            if d(v) > d(u) ⊕_B c(uv) :
                d(v) ← d(u) ⊕_B c(uv)
                π(v) ← u
                if v ∉ Q :
                    Q.InsertLast(v)
    return d
```

```
e-Dijkstra(G = (V, A, c), p, B, s):
    for u ∈ V do
        d(u) ← ∞
        π(u) ← null
    d(s) ← 0
    H ← min-heap()
    H.insert(s, −p(s))
    while H ≠ ∅ :
        v ← H.delete-min()
        for uv ∈ A do
            if d(v) > d(u) ⊕_B c(uv) :
                d(v) ← d(u) ⊕_B c(uv)
                π(v) ← u
                if v ∉ H :
                    H.insert(u, d(v)−p(v))
                else:
                    H.decrease-key(u, d(v)−p(v))
    return d
```

**Figure 3** Energetic variants of the Bellman-Ford and Dijkstra algorithms.

the algorithm does the appropriate *decrease-key* operation on the heap. If all arc costs are non-negative, the algorithm deletes each vertex from $H$ at most once, and when a vertex $v$ is deleted from $H$, $d(v)$ is the minimum energetic cost of a path from $s$ to $v$. The proof of correctness mimics that of the standard Dijsktra algorithm. The algorithm does at most $n$ heap insertions, at most $n$ heap deletions, and at most $m$ decrease-key operations. If the heap is a Fibonacci heap [16] or equally efficient data structure, e.g., [20], the total running time is $O(m + n \log n)$. In fact, the algorithm is identical to the standard algorithm with $d(v)$ values greater than $B$ replaced by $\infty$.

More interesting is that if arc costs can be negative, but there are no negative cycles, we can use a variant of the $A^*$ search algorithm, which is a modification of Dijkstra's algorithm, to solve the single-source minimum energetic paths problem in $O(m + n \log n)$ time, provided that we have a *valid potential function* $p : V \to \mathbb{R}$. A potential $p$ is *valid* if $c(uv) + p(u) - p(v) \geq 0$ for every arc $uv \in A$. It is well-known that a valid potential function exists if and only if the graph contains no negative cycles.

The $A^*$ search algorithm is almost identical to Dijkstra's algorithm. The only difference is that the *key* of vertex $v$ in the heap is $d(v) - p(v)$, and not just $d(v)$, where $p$ is a valid potential function. In the original setting of the $A^*$ search heuristic, $-p(v)$ is an estimate of the distance from $v$ to the destination $t$. The correctness of the algorithm only requires, however, that $p$ is a valid potential function. If $p$ is valid, the $A^*$ algorithm deletes each vertex $v$ from the heap at most once, and when $v$ is deleted, $d(v) = \delta_B(s, v)$, the energetic cost of traveling from $s$ to $v$.

An energetic version of the $A^*$ is obtained simply by replacing $+$ by $\oplus_B$ in relaxations. We assume the algorithm is given a potential $p$ that is valid for $+$, not $\oplus_B$. The algorithm begins with $d(s) = 0$ and $d(v) = \infty$ for each vertex $v \in V \setminus \{s\}$, and $H$ containing $s$. The key of a vertex $v$ in $H$ is $d(v) - p(v)$. The algorithm repeats the following step until $H$ is empty:

Scan a vertex: Delete from $H$ a vertex $u$ with minimum key $d(u) - p(u)$. For each arc $uv$, if $d(u) \oplus_B c(uv) < d(v)$, *relax* $uv$: Set $d(v) \leftarrow d(u) \oplus_B c(uv)$; $\pi(v) \leftarrow u$; add $v$ to $H$ with key $d(v) - p(v)$ if $v \notin H$, or decrease the key of $v$ to $d(v) - p(v)$ if $v \in H$.

Pseudocode of the resulting algorithm, which we call *e-Dijkstra*, is given on the right of Figure 3. The main step towards establishing the correctness of *e-Dijkstra* is the following:

▶ **Lemma 4.1.** *If $p$ is a valid potential then e-Dijkstra maintains the following invariant: if $u$ has been deleted from $H$ while $v$ has not been deleted from $H$ yet, then $d(u)-p(u) \le d(v)-p(v)$. As a consequence, each vertex $u$ is inserted and deleted from $H$ at most once.*

**Proof.** We prove the lemma by induction on the number of heap operations. The lemma is true initially, as no vertex was deleted from $H$ yet. Suppose it is true just before $u$ is deleted from $H$. Since $d(u) - p(u)$ is *minimum* among all $u \in H$, and since $d(v) = \infty$ for all vertices not yet inserted into $H$, the invariant holds just after $u$ is deleted from $H$. By the induction hypothesis, $d(u) - p(u)$ is now *maximum* over all $u'$ already deleted from $H$. Suppose the invariant holds just before the relaxation of an arc $uv$. Just after the relaxation, $d(v) = d(u) \oplus_B c(uv) \ge d(u) + c(uv)$. Hence

$$d(v) - p(v) \;\ge\; d(u) + c(uv) - p(v) \;\ge\; d(u) - p(u)\ ,$$

where the last inequality follows by the validity of $p$. Since the relaxation strictly decreased $d(v)$, it follows that $v$ could not have already been deleted from $H$, since it would violate the claim that $d(u) + p(u)$ is maximum over all vertices already deleted from $H$. Thus, $v$ is either in $H$ or was not inserted into $H$ yet. Decreasing the key of $v$ to $d(v) - p(v)$, or inserting $v$ into $H$ with this key, does not violate the invariant. ◀

The proof of Lemma 4.1 is the same as the proof of the corresponding lemma for the standard version of $A^*$ except for the use of the inequality $x \oplus_B y \ge x + y$. Using Lemma 4.1 we can easily prove the correctness of the algorithm.

▶ **Theorem 4.2.** *If $G = (V, A, c)$ has no negative cycles and $p$ is a valid potential for $G$, then e-Dijkstra finds minimum energetic paths from $s$ to all vertices in $O(m + n \log n)$ time.*

**Proof.** When a vertex $u$ is removed from $H$, all outgoing arcs $uv$ are scanned and all appropriate relax operations are performed. By Lemma 4.1, $d(u)$ will not be changed again. Thus, when the algorithm terminates $d(v) \le d(u) \oplus_B c(uv)$ for every arc $uv \in A$. By Corollary 3.2, we have $d(v) = \delta_B(s, v)$, for every $v \in V$. As in the proof of Theorem 3.1 we get that the $\pi$ values describe a tree of minimum energetic paths from $s$ to all vertices that can be reached from $s$.

The algorithm performs at most $n$ heap insertions, at most $n$ heap deletions, and at most $m$ decrease-key operations. With an efficient heap implementation the total running time is $O(m + n \log n)$. ◀

To obtain a valid potential function we can use any standard shortest path algorithm: If $s$ is an arbitrary source from which all vertices are reachable, there are no negative cycles, and $p(v) = \delta(s, v)$, where $\delta(s, v)$ is the standard distance from $s$ to $v$, then $c(uv) + p(u) - p(v) \ge 0$, for every arc $uv$, by the triangle inequality. (If there is no such vertex $s$ in the graph, add a new vertex $s$ and connect it with zero-cost arcs to all other vertices.)

Thus we can compute minimum energetic paths from $k$ sources in $O(m + n \log n)$ time per source plus the time to solve one standard single-source shortest path problem with the given arc costs. The extra time needed for this preprocessing is $O(mn)$ if we use Bellman-Ford,

or $O(m \log^2 n \log(nW) \log\log n)$ if all arc costs are integral and we use the algorithm of Bringmann et al. [8]. (The faster algorithm is helpful only if $k = \Omega(\log^2 n \log(nW) \log\log n)$, where $k$ is the number of sources.)

▶ **Corollary 4.3.** *The all-pairs minimum energetic paths problem on a graph $G = (V, A, c)$ with no negative cycles can be solved in $O(mn + n^2 \log n)$ time.*

The resulting all-pairs algorithm is very similar to Johnson's [22] algorithm for the standard all-pairs shortest paths problem.

## 5 Finding minimum paths in the presence of negative cycles

In this section we describe an algorithm *e-Negative*$(G, B, s)$ that finds minimum energetic costs from a source vertex $s \in V$ to all other vertices in a directed graph $G = (V, A, c)$ that may contain negative cycles. It is not difficult to extend the algorithm to return a succinct description of the minimum energetic paths. We assume that $s$ has no incoming arcs. (Incoming arcs of $s$ are not useful as we start from $s$ with a full battery.)

Algorithm *e-Negative* maintains a set $R$ of reachable vertices that were already processed, a set of reachable vertices $Q$ that are waiting to be processed, and a set $Y \subseteq R$ of exits. Initially $R = \emptyset$, $Q = \{s\}$ and $Y = \emptyset$. Let $G_{R,Y}$ be the graph obtained from $G$ by removing the outgoing arcs of vertices not in $R$, removing the incoming arcs of vertices in $Y$, and for every $y \in Y$, adding a 0-cost arc $sy$. (Note that vertices in $V \setminus R$ may have incoming arcs in $G_{R,Y}$, but no outgoing arcs.) We may assume that the vertex set of $G_{R,Y}$ is $V_{R,Y} = R \cup N(R)$, where $N(R)$ are the out-neighbors of the vertices of $R$. We also have $Q \subseteq N(R)$.) The algorithm maintains the invariant that $G_{R,Y}$ has no negative cycles and $p : V_{R,Y} \to \mathbb{R}$ is a valid potential function for it, and that all vertices in $R \cup Q$ can be reached when starting from $s$ with a full battery.

The algorithm is composed of rounds. In each round the algorithm removes a vertex $u \in Q$ and processes it, i.e., adds it to the set $R$. If $G_{R\cup\{u\},Y}$ does not contain negative cycles, all we need to do is find a valid potential function for $G_{R\cup\{u\},Y}$ and update the set $Q$ of vertices reachable in $G_{R\cup\{u\},Y}$ when starting from $s$ with a full battery.

To check whether $G_{R\cup\{u\},Y}$ contains a negative cycle we construct a graph $\bar{G} = \bar{G}_{R,Y,u}$ as follows. The graph is obtained by starting from $G_{R,Y}$, adding a new source vertex $\bar{u}$, and for every $uv \in A$, adding an arc $\bar{u}v$ with $c(\bar{u}v) = c(uv)$. We also remove $s$ and its outgoing arcs. (Note that $u \notin R$ has no outgoing arcs in $\bar{G}$.) The new graph $\bar{G}$ does not contain negative cycles. The function $p$ is a valid potential function for $\bar{G}$ if we let $p(\bar{u}) = \max_{uv\in A}(p(v) - c(uv))$.[2] All negative cycles in $G_{R\cup\{u\},Y}$ must pass through $u$. Thus, $G_{R\cup\{u\},Y}$ contains a negative cycle if and only if $\delta_{\bar{G}}(\bar{u}, u) < 0$.

We thus run Dijkstra on $\bar{G}$ staring from the source $\bar{u}$ using the potential function $p$. If $\delta_{\bar{G}}(\bar{u}, u) < 0$ then a shortest path from $\bar{u}$ to $u$ becomes a negative cycle $C$ in $G_{R\cup\{u\},Y}$ when we replace $\bar{u}$ with $u$. (Any path of negative cost from $\bar{u}$ to $u$ will do. We can thus stop the algorithm as soon as such a path is discovered.) In $O(|C|)$ time we find an exit $y$ on $C$ and add it to $Y$. This removes the incoming arcs of $y$ from $G_{R\cup\{u\},Y}$ and adds a 0-cost arc $sy$. We update the graph $\bar{G}$ accordingly, i.e., remove the incoming arcs of $y$, and run Dijkstra again. (Note that $p$ is still a valid potential function for $\bar{G}$, since $s$ is not included in it.)

[2] This works as $\bar{u}$ has no incoming arcs. Equivalently we can use the fact that Dijkstra's algorithm works correctly even if the reduced costs of some of the outgoing arcs of the source are negative, which follows as $p(\bar{u})$ does not really affect the running of the algorithm. (It only affects the key of $\bar{u}$ when it is alone in the heap.

When no more negative cycles are found, the graph $G_{R\cup\{u\},Y'}$, where $Y'$ is the new set of exits, does not contain negative cycles. We still need to find a valid potential function for it. To do this it is enough to find distances from $s$ to all other vertices. Let $\delta(s,v)$ be the distance from $s$ to $v$ in $G_{R\cup\{u\},Y'}$. Let $\delta'(s,v)$ be the distance from $s$ to $v$ in $G_{R,Y'}$. Finally, let $\delta''(\bar{u},v)$ be the distance from $\bar{u}$ to $v$ in $\bar{G}_{R,Y',u}$. Clearly, $\delta(s,v) = \min\{\delta'(s,v), \delta'(s,u) + \delta''(\bar{u},v)\}$, since each shortest path in $G_{R\cup\{u\},Y'}$ either does not pass through $u$, in which case its length is $\delta'(s,v)$, or it does pass through $u$ in which case its length is $\delta'(s,u) + \delta''(\bar{u},v)$.

To find the distances $\delta'(s,v)$ we simply run Dijkstra on $G_{R,Y'}$ using the potential function $p$, after we set $p(s) = \max_{sv\in A}(p(v) - c(sv))$. The distances $\delta''(\bar{u},v)$ were already computed. Thus we can easily compute the distances $\delta(s,v)$ in $G_{R\cup\{u\},Y'}$ and then set $p(v) = \delta(s,v)$, for every $v \in V_{R\cup\{u\},Y'}$.

Given the newly computed potential function $p$, we can now run *e-Dijkstra* on $G_{R\cup\{u\},Y'}$ to compute energetic costs and the new set of vertices $Q'$ that are reachable from $s$ but are not in $R\cup\{u\}$. The algorithm then lets $R \leftarrow R\cup\{u\}$, $Y \leftarrow Y'$ and $Q \leftarrow Q'$. This completes the round. If $Q = \emptyset$ the algorithm terminates. Otherwise it proceeds to the next round, processing the next vertex from $Q$.

We claim that the energetic costs computed by the last *e-Dijkstra*, which are also the values returned by *e-Negative*, are exactly the energetic costs $\delta_B(s,v)$ in the input graph $G = (V,A,c)$. (If $v \notin R$, then $\delta_B(s,v) = \infty$.)

▶ **Theorem 5.1.** *Algorithm e-Negative$(G,B,s)$ finds minimum energetic costs in a graph $G = (V,A,c)$ that may contain negative cycles when the capacity of the battery is $B$. Its running time is $O(mn + n^2 \log n)$.*

**Proof.** The correctness of the algorithm follows from the explanations above combined with a few simple observations. It follows by induction that at the beginning of each round $G_{R,Y}$ contains no negative cycles, $p$ is a valid potential function for it, and all vertices in $R\cup Q$ can be reached when starting from $s$ with a full battery.

In each round, the algorithm moves a vertex $u$ from $Q$ to $R$. To do this, it repeatedly finds negative cycles in the graph $G_{R\cup\{u\},Y}$. All such negative cycles must pass through $u$. As explained, there is such a negative cycle in $G_{R\cup\{u\},Y}$ if and only if $\delta_{\bar{G}}(\bar{u},u) < 0$, where $\bar{G} = \bar{G}_{R,Y,u}$. A negative path from $\bar{u}$ to $u$ corresponds to a negative cycle $C$, since $\bar{u}$ and $u$ represent the same vertex. If a negative cycle $C$ is found, an exit $y$ on $C$ is identified and added to $Y$. (Note that this removes arcs from $\bar{G}$ since the incoming arcs of $y$ are removed.) The arc $sy$ is added to $G_{R\cup\{u\},Y\cup\{y\}}$ but not to $\bar{G} = \bar{G}_{R,Y\cup\{y\},u}$.

We next argue that $\delta_B(s,y) = 0$, for every $y \in Y$. Indeed, each vertex $y$ added to $Y$ is an exit on a negative cycle $C$ all whose vertices can be reached when starting from $s$ with a full battery. In particular, the entry $x$ on $C$ corresponding to $y$ can be reached, and by the definition of entry-exit pairs, $y$ can be reached with full battery, i.e., $\delta_B(s,y) = 0$. This also justifies the addition of the 0-cost arc $sy$.

Let $\delta'_B(s,v)$ be the energetic costs computed by the last call to *e-Dijkstra* on the final $G_{R,Y}$. We assume that $\delta'(s,v) = \infty$ for every $v \notin R$. Let $\delta_B(s,v)$ be the energetic costs in the input graph $G$. It is easy to see that $\delta_B(s,v) \le \delta'(s,v)$, for every $v \in V$. This follows as $G_{R,Y}$, without the arcs $\{sy \mid y \in Y\}$, is a subgraph of $G$, and the addition of the arcs $\{sy \mid y \in Y\}$, as argued, does not change the energetic costs.

We next show that when the algorithm terminates we have $\delta_B(s,v) = \delta'_B(s,v)$, for every $v \in V$. Suppose, for the sake of contradiction, that there is a vertex $v$ for which $\delta_B(s,v) < \delta'_B(s,v)$. Let $P$ be a minimum energetic path from $s$ to $v$ in $G$. If $P$ passes through a vertex of $Y$, let $y$ be the last vertex from $Y$ on $P$ and let $P'$ be path composed

of the arc $sy$ followed by the portion of $P$ from the last occurrence of $y$ on $P$ to $v$. Since $\delta_B(s,v) < \delta'_B(s,v)$, it follows that $P'$ is not a path in $G_{R,Y}$. Thus, $P'$ must contain a vertex not in $R$. Let $u$ be the first vertex not in $R$ on $P'$. It follows that $\delta'_B(s,u) < \infty$, since the portion of $P'$ from $s$ to $u$ is also a path in $G_{R,Y}$. Since $u \notin R$, we must have $u \in Q$ and the algorithm should not have terminated.

The algorithm performs at most $2n$ calls to Dijkstra, since following each such call either an exit is added to $Y$ or a vertex is added to $R$. The algorithm performs at most $n$ calls to *e-Dijkstra*. Thus, the running time of the algorithm is $O(mn + n^2 \log n)$. ◀

# 6 A faster all-pairs algorithm for large batteries

In this section we obtain an $O(mn + n^2 \log n)$-time algorithm for the all-pairs energetic cost problem, in graphs that may contain negative cycles, when the capacity of the battery $B$ is sufficiently large relative to the arc costs in the graph. The initial charge $b$ may be arbitrary. More specifically, we assume that $B \geq 3nM$, where $M = \max_{uv \in A} |c(uv)|$.

Recall that $\delta_{B,b}(s,t)$ is the energetic cost of getting from $s$ to $t$, starting from $s$ with a charge $b$, where $0 \leq b \leq B$, when the capacity of the battery is $B$. The energetic cost is the difference between the initial charge, in this case $b$, and the final charge. Recall that $\delta_{B,b}(s,t) \in [b - B, b]$. (In particular, when $b < B$, the energetic cost may be negative.)

We started Section 2 with a simple reduction from the computation of $\delta_{B,b}(s,t)$ to that of $\delta_B(s,t) = \delta_{B,B}(s,t)$. The reduction introduces an arc of cost $B - b$, which may be much larger than $M$, so we cannot use it when the battery capacity is large. To make our results more general we work directly with $\delta_{B,b}(s,t)$.

The improved algorithm is obtained by using a preprocessing step, described in Section 6.1, that finds sets of entries or exits that *hit* all negative cycles. When the battery is large, such sets can be used to efficiently solve the single-source problem, from any source, in $O(m + n \log n)$ time.

We first describe, in Section 6.2, an efficient algorithm when $b \geq nM$. We then use this algorithm in Section 6.3 to obtain an efficient algorithm when $B \geq 3nM$ and $b$ is arbitrary.

## 6.1 Finding sets of entries or exits that hit all negative cycles

A vertex $x$ is an entry if it is an entry on some negative cycle. Similarly, a vertex $y$ is an exit if it is an exit on some negative cycle.

A set of vertices $Z \subset V$ is said to hit all negative cycles in a graph $G$ if there are no negative cycles in the graph $G \setminus Z$, or equivalently in the graphs $G \setminus in(Z)$ or $G \setminus out(Z)$, where $G \setminus Z$ is the graph obtained by removing all the vertices of $Z$ from $G$ and $G \setminus in(Z)$ and $G \setminus out(Z)$ are the graphs obtained just by removing the incoming or outgoing arcs, respectively, of the vertices in $Z$.

We are interested in hitting all negative cycles with either a set of entries, or a set of exits. We show that this can be done efficiently.

▶ **Lemma 6.1.** *A set $Y \subseteq V$ of exits that hit all negative cycles in $G$, and a valid potential function $p$ for $G \setminus in(Y)$, can be found in $O(mn + n^2 \log n)$ time.*

**Proof.** Add to $G$ an auxiliary source vertex $\bar{s}$ and connect it with 0-cost arcs to all other vertices of $G$. Running *e-Negative* of Section 5 on the resulting graph will construct a set $Y$ of exits that hit all negative cycles in $G$ and a valid potential function $p$. ◀

By a slight adaptation of the algorithm, i.e., finding an entry on each negative cycle found and removing its outgoing arcs, we can also get:

▶ **Lemma 6.2.** *A set $X \subseteq V$ of entries that hit all negative cycles in $G$, and a valid potential function $p$ for $G \setminus out(X)$, can be found in $O(mn + n^2 \log n)$ time.*

## 6.2 An algorithm for large initial charges

We assume that $nM \leq b \leq B$, i.e., the initial charge is sufficient to traverse any path of at most $n$ arcs. Let $u \rightsquigarrow v$ denote that there is a directed path from $u$ to $v$ in the graph.

Suppose that $C$ is a negative cycle in $G$ and that $(x, y)$ is an entry-exit pair on it. Suppose that $s$ is the current source. Since $x$ and $y$ are on a cycle we clearly have $s \rightsquigarrow x$ if and only if $s \rightsquigarrow y$. Furthermore, if $s \rightsquigarrow y$ then there is also a simple path from $s$ to $x$, i.e., a path that uses at most $n - 1$ arcs. Since we start from $s$ with a sufficiently large initial charge, we can reach $x$ and eventually, by the definition of entry-exit pairs, reach $y$ with a fully charged battery. This justifies the following algorithm.

Find a set $Y$ of exits that hit all negative cycles and find a valid potential function $p$ for $G \setminus in(Y)$. For any given source $s$, find the set $Y_s \subseteq Y$ of exits reachable from $s$. (This can be easily done in linear time.) Next, construct a graph $G_s$ obtained from $G \setminus in(Y)$ by adding arcs of cost $b - B$ from $s$ to every $y \in Y_s$. (These arcs ensure that we reach $y$ with a fully charged battery without needing to use the incoming arcs of $y$.) Run *e-Dijkstra* on $G_s$ using $p$, after suitably adjusting the potential of $s$. (We need a slight modification of *e-Dijkstra* that works when starting from $s$ with an initial charge $b$. Alternatively, we can add an auxiliary source $\bar{s}$, add an arc $\bar{s}s$ of cost $B - b$, and run *e-Dijkstra* from $\bar{s}$.) This takes only $O(m + n \log n)$ time per vertex, giving an $O(mn + n^2 \log n)$-time algorithm for the all-pairs problem.

## 6.3 An algorithm for large batteries

We now describe an algorithm for $3nM \leq B$, and any value of $b$. If $nM \leq b$, we can use the algorithm of the previous section. We can thus assume that $b \leq nM$ and thus $B - b \geq 2nM$, i.e., the battery is initially far from being fully charged.

Start by using the algorithm from Section 6.1 to find a set $X$ of entries that hit all negative cycles of $G$.

Let $P$ be a minimum energetic path from $s$ to $t$. We may assume that all cycles on $P$, if any, are negative, since otherwise they can be removed without increasing the energetic cost. If $P$ does not pass through any vertex of $X$, then it is also a path in $G \setminus out(X)$. Otherwise, let $x \in X$ be the first entry appearing on $P$. Suppose that $x$ is an entry of a negative cycle $C$ and that $y$ is a corresponding exit on $C$. In general, the path $P$ may not pass through $y$. This is one of the main difficulties that algorithm *e-Negative* had to deal with. However, when $B - b \geq 2nM$, we show that there must exist a minimum energetic path $P'$ from $s$ to $t$ that passes through both $x$ and $y$. We can further assume that during the last visit of $y$, the battery is fully charged.

The path $P$ must reach the first entry $x \in X$ after traversing at most $n - 1$ arcs. If $P$ is simple, this is obviously true. Otherwise, a cycle $C'$ must be formed after traversing at most $n$ arcs. By definition, there is an entry $x \in C' \cap X$. (Note that $x$ is not necessarily an entry of $C'$, but it is an entry of some negative cycle $C$ with a corresponding exit $y$.)

If $b_x$ is the charge in the battery when reaching the first entry $x$ on $P$, then $B - b_x \geq nM$. (The additional charge gained by traversing the portion of $P$ from $s$ to $x$ is at most $nM$.) Since $x$ is an entry of $C$, we can traverse $C$ and get back to $x$, passing $y$ along the way, with a charge that is at least $b_x$. (Note that it is important here that the battery is not close to being fully charged when starting from $x$, otherwise the claim is not necessarily true.) This gives us the path $P'$.

This suggests the following algorithm. Start, as mentioned, by finding a set $X$ of entries that hit all negative cycles and a potential function $p$ for $G \setminus out(X)$.

For each vertex $s \in V$, run *e-Dijkstra* on $G \setminus out(X)$ starting from $s$ using the potential function $p$. This finds minimum energetic paths from $s$ that do not pass through $X$. It also finds the set $X_s$ of entries that can be reached when staring from $s$ with an initial charge of $b$. Let $Y_s$ be the set of exits that correspond to the entries in $X_s$. Construct a new graph $G_s$ as follows. Remove from $G$ all the outgoing arcs of $s$. For each $y \in Y_s$ add a 0-arc from $s$ to $y$. Now run the algorithm of Section 6.2, staring from $s$ with a fully charged battery. (Assuming that a set $Y$ of exits that hit all negative cycles of $G$ was already precomputed.) Subtract $B - b$ from the results obtained, to adjust for the fact that the initial charge is actually $b$ and not $B$, and take the minimum with the results returned by *e-Dijkstra* call.

The computation of the hitting sets $X$ and $Y$ takes $O(mn + n^2 \log n)$. For every source we need only $O(m + n \log n)$ time. The total time of the algorithm is thus $O(mn + n^2 \log n)$.

## 7 Minimum initial charges and maximum final charges

To end the paper, we consider two problems that are closely related to the minimum energetic paths problem. Let $G = (V, A, c)$ be a graph with no (traversable) negative cycles and let $B$ be the capacity of the battery. For two vertices $s, t \in V$, we let $\alpha_B(s, t)$ be the *maximum final charge* with which it is possible to reach $t$ when starting at $s$ with a full battery, or $-\infty$, if it is not possible to travel from $s$ to $t$. We also let $\beta_B(s, t)$ be the *minimum initial charge* required at $s$ for getting to $t$, or $\infty$, if no initial charge (of at most $B$) is sufficient.

The maximum final charge problem is not really a new problem as $\alpha_B(s, t) = B - \delta_B(s, t)$. As noted in the introduction, $\beta_B(s, t)$ is the smallest $b$ such that $\delta_{B,b}(s, t) \leq b$, or equivalently $\delta_{B,b}(s, t) < \infty$, if there is such a $b$. Thus, if $B$ and all arc costs are integral, then we can find $\beta_B(s, t)$, for a specific pair $s$ and $t$, by a binary search.

There is, however, a more interesting relation between the minimum initial-charge problem and the minimum energetic cost problem. Namely, $\beta_B(s, t)$ is equal to $\delta_B^{\overleftarrow{G}}(t, s)$ the energetic cost of traveling from $t$ to $s$ in the *reversed graph* $\overleftarrow{G}$, the graph obtained by reversing all the arcs in the graph $G$ and retaining all arc costs. This relation follows easily from the following lemma, analogous to Lemma 2.2, whose simple proof is omitted. For a path $P$ from $s$ to $t$, let $b_B(P)$ be the minimum initial charge at $s$ with which the path $P$ can be traversed.

▶ **Lemma 7.1.** *Let $P = u_0 u_1 \dots u_k$ be a directed path, let $P' = u_1 \dots u_k$, and let $c_i = c(u_{i-1}u_i)$, for $i = 1, \dots, k$. Let $B$ be the capacity of the battery. If $k = 0$ then $b_B(P) = 0$. If $k > 0$ then*

$$b_B(P) \;=\; c_1 \oplus_B b_B(P') \;=\; c_1 \oplus (c_2 \oplus (\cdots \oplus (c_{k-1} \oplus (c_k \oplus 0)) \cdots)) \,.$$

▶ **Corollary 7.2.** *For every $s, t \in V$, $\beta_B(s, t) = \delta_B^{\overleftarrow{G}}(t, s)$.*

As immediate corollaries, it follows that we can solve the single-target version of the minimum initial-charge paths problem in $O(m+n \log n)$ time, if we are given a valid potential, and the all-pairs version of the problem in $O(mn + n^2 \log n)$.

## 8 Concluding remarks and open problems

We have presented a clear definition of the minimum energetic paths problem, which is a strict extension of the standard shortest paths problem, and explained its relation to two other related problems: minimum initial-charge paths and maximum final-charge paths. We have also presented efficient algorithms for the minimum energetic paths problem in three different settings.

When there are no negative cycles in the graph, the minimum energetic paths problem can be solved using relatively simple adaptations of the classical Bellman-Ford and Dijkstra algorithms. We present simple descriptions and simple correctness proofs of these algorithms. In particular, we have obtained an $O(mn)$-time algorithm for the single-source version, when arc costs may be negative but there are no negative cycles, and an $O(mn + n^2 \log n)$-time algorithm for the all-pairs version.

An interesting feature of the minimum energetic paths problem is that it is well defined even if the graph contains negative cycles. Furthermore, minimum energetic costs are always obtained by finite length, though not necessarily simple, minimum energetic paths. (This is not the case for the standard shortest paths problem.) Using new algorithmic techniques, we have obtained an $O(mn + n^2 \log n)$-time algorithm for the single-source version of the problem. Our best algorithm for the all-pairs version runs the single-source algorithm from each vertex, yielding a running time of $O(mn^2 + n^3 \log n)$.

We have obtained a more efficient algorithm for the all-pairs version when the capacity of battery is sufficiently large, i.e., $B \geq 3nM$, where $M = \max_{uv \in A} |c(uv)|$ is the maximum absolute value of an arc cost. The running time of the improved algorithm is $O(mn+n^2 \log n)$.

The obvious open problems are whether any of our time bounds can be improved. In particular, is it possible to get an $O(mn)$-time algorithm for the single-source version when the graph may contain negative cycles? Is there an $O(n^3)$-time algorithm for the all-pairs version when the graph may contain negative cycles?

Another interesting problem is whether the new techniques of Bernstein et al. [5] and Bringmann et al. [8], or the older technique of Goldberg [19] can be used to obtain an improved algorithm for the single-source version of the minimum energetic paths problem, when negative cycles may be present in the graph.

Finally, as mentioned, the single-target version of the minimum energetic paths problem is not equivalent to the single-source version. Currently, our fastest algorithms for the single-target version actually solve the all-pairs version. Is there a faster solution? The fact that there may not be a tree of minimum energetic paths to a given target may indicate that the single-target version is harder than the single-source version.

## References


1 Shaull Almagor, Nathann Cohen, Guillermo A. Pérez, Mahsa Shirmohammadi, and James Worrell. Coverability in 1-vass with disequality tests. In *Proc. of 31st CONCUR*, volume 171 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.38.

2 Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The shortest path problem revisited: Optimal routing for electric vehicles. *KI*, 6359:309–316, 2010.

3 Moritz Baum, Julian Dibbelt, Thomas Pajor, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Energy-optimal routes for battery electric vehicles. *Algorithmica*, 82:1490–1546, 2020.

4 Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

5 Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 600–611. IEEE, 2022. doi:10.1109/FOCS54457.2022.00063.

6 Lubos Brim and Jakub Chaloupka. Using strategy improvement to stay alive. *Int. J. Found. Comput. Sci.*, 23(3):585–608, 2012. doi:10.1142/S0129054112400291.

7 Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.

8 Karl Bringmann, Alejandro Cassis, and Nick Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! *CoRR*, abs/2304.05279, 2023. doi:10.48550/arXiv.2304.05279.

9 Shiri Chechik, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick. Bottleneck Paths and Trees and Deterministic Graphical Games. In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, pages 27:1–27:13, 2016.

10 Boris V. Cherkassky, Loukas Georgiadis, Andrew V. Goldberg, Robert E. Tarjan, and Renato F. Werneck. Shortest-path feasibility algorithms: An experimental evaluation. *ACM J. Exp. Algorithmics*, 14, 2009. doi:10.1145/1498698.1537602.

11 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

12 Dani Dorfman, Haim Kaplan, and Uri Zwick. A faster deterministic exponential time algorithm for energy games and mean payoff games. In *Proc. of 46th ICALP*, volume 132 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.114.

13 Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. of 20th SODA*, pages 384–391, 2009. doi:10.1145/1496770.1496813.

14 Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large networks. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3637.

15 Lester R. Ford. Network flow theory. Technical Report Paper P-923, RAND Corporation, Santa Monica, California, 1956.

16 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

17 Harold N. Gabow and Robert E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.

18 Judith F. Gilsinn and Christoph Witzgall. A performance comparison of labeling algorithms for calculating shortest path trees. Technical Report 772, US National Bureau of Standards, 1973.

19 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

20 Thomas Dueholm Hansen, Haim Kaplan, Robert E. Tarjan, and Uri Zwick. Hollow heaps. *ACM Trans. Algorithms*, 13(3):42:1–42:27, 2017. doi:10.1145/3093240.

21 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

22 Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977. doi:10.1145/321992.321993.

23 Samir Khuller, Azarakhsh Malekian, and Julián Mestre. To fill or not to fill: The gas station problem. *ACM Transactions on Algorithms (TALG)*, 7(3):1–16, 2011.

24 David A Klarner, editor. *The mathematical gardner*. Wadsworth International, 1982.

25 Donald E. Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977. doi:10.1016/0020-0190(77)90002-3.

26 Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Wegrzycki. Coverability in VASS revisited: Improving rackoff's bound to obtain conditional optimality. *CoRR*, abs/2305.01581, 2023. doi:10.48550/arXiv.2305.01581.

27 Daniel J. Lehmann. Algebraic structures for transitive closure. *Theor. Comput. Sci.*, 4(1):59–76, 1977. doi:10.1016/0304-3975(77)90056-1.

28 László Lovász. *Combinatorial problems and exercises*, volume 361. American Mathematical Soc., 2007.

29 Omid Madani, Mikkel Thorup, and Uri Zwick. Discounted deterministic markov decision processes and discounted all-pairs shortest paths. *ACM Trans. Algorithms*, 6(2):33:1–33:25, 2010. doi:10.1145/1721837.1721849.

30 Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350, 2002. doi:10.25596/jalc-2002-321.

31 Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theor. Comput. Sci.*, 312(1):47–74, 2004. doi:10.1016/S0304-3975(03)00402-X.

32 Robert E. Tarjan. Shortest paths. Technical report, AT&T Bell Laboratories, 1981.

33 Robert E. Tarjan. A unified approach to path problems. *Journal of the ACM*, 28(3):577–593, 1981. doi:10.1145/322261.322272.

34 Robert E. Tarjan. *Data structures and network algorithms*. SIAM, 1983.

35 Virginia Vassilevska. Nondecreasing paths in a weighted graph or: how to optimally read a train schedule. In *Proc. of 19th SODA*, pages 465–472, 2008. doi:10.1145/1347082.1347133.

36 Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory Comput.*, 5(1):173–189, 2009. doi:10.4086/toc.2009.v005a009.

37 Peter Winkler. *Mathematical puzzles: a connoisseur's collection*. A K Peters, 2004.

38 Uri Zwick. Exact and approximate distances in graphs - A survey. In *Proc. of 9th ESA*, volume 2161 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2001. doi:10.1007/3-540-44676-1_3.

# Evaluating Restricted First-Order Counting Properties on Nowhere Dense Classes and Beyond

**Jan Dreier**
TU Wien, Austria

**Daniel Mock**
RWTH Aachen University, Germany

**Peter Rossmanith**
RWTH Aachen University, Germany

## Abstract

It is known that first-order logic with some counting extensions can be efficiently evaluated on graph classes with bounded expansion, where depth-$r$ minors have constant density. More precisely, the formulas are $\exists x_1 \dots x_k \# y\, \varphi(x_1, \dots, x_k, y) > N$, where $\varphi$ is an FO-formula. If $\varphi$ is quantifier-free, we can extend this result to *nowhere dense* graph classes with an almost linear FPT run time. Lifting this result further to slightly more general graph classes, namely almost nowhere dense classes, where the size of depth-$r$ clique minors is subpolynomial, is impossible unless FPT = W[1]. On the other hand, in almost nowhere dense classes we can approximate such counting formulas with a small additive error. Note those counting formulas are contained in $\mathrm{FOC}(\{>\})$ but not $\mathrm{FOC}_1(\mathbf{P})$.

In particular, it follows that partial covering problems, such as partial dominating set, have fixed parameter algorithms on nowhere dense graph classes with almost linear running time.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Logic; Theory of computation → Computational complexity and cryptography; Mathematics of computing → Graph theory

**Keywords and phrases** nowhere dense, sparsity, counting logic, dominating set, FPT

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.43

**Related Version** *Full Version*: https://arxiv.org/abs/2307.01832

**Funding** Supported by the German Science Foundation (DFG), grant no. DFG-927/15-2.

## 1 Introduction

First-order logic can be used to express algorithmic problems. FO-model checking on certain classes of structures is therefore a meta-algorithm, which solves many problems at the same time. For example, the three classical problems that started the research on parameterized complexity are all FO-expressible: Vertex Cover, Independent Set, and Dominating Set [6, 7]. Dominating Set with the natural parameter – the size of the minimal dominating set – is W[2]-complete on general graphs, but fixed parameter tractable (fpt) on many special graph classes. The study of *sparsity*, initiated by Nešetřil and Ossona de Mendez, has led to the concept of *bounded expansion* and *nowhere dense* graph classes [21]. They generalize many well-known notions of sparsity, such as bounded degree, planarity, bounded genus, bounded treewidth, (topological) minor-closed, etc. and have led to quite general algorithmic results [23, 11, 4, 10]. Most notably, Grohe, Kreutzer, and Siebertz showed that FO-model checking is fpt on nowhere dense graph classes [15]. This shows, e.g., that dominating set is fpt on nowhere dense graphs, a result that was already known: Dawar and Kreutzer were able to find a specific algorithm several years earlier [5] that solves generalizations of the dominating set problem. All of them are FO-expressible, which shows how strong meta-algorithms are.


Partial dominating set, also called $t$-dominating set, is another generalization of dominating set: The input is a graph $G$ and two numbers $k$ and $t$. The question is, whether $G$ contains $k$ vertices that dominate at least $t$ vertices. The parameter is $k$, as in the classical dominating set problem. (If you choose $t$ as the parameter – which also makes sense – the problem becomes fixed-parameter tractable even on general graphs [18].) The length of an FO-formula expressing the existence of a partial dominating depends on $t$, which is not bounded by any function of $k$ and therefore all the results on first-order model checking do not help when we parameterize by $k$ only. Golovach and Villanger showed that partial dominating set remains hard on degenerate graphs [13], while Amini, Fomin, and Saurabh have shown that partial dominating set is fixed-parameter tractable in minor-closed graph classes, which generalized earlier positive results [1]. Very recently, this was improved to graph classes with bounded-expansion, while simultaneously using only linear fpt time instead of polynomial fpt time, i.e, the running time is now only $f(k)n$ [8].

This result was achieved by another meta-theorem for the counting logic FOC($\{>\}$) on classes of bounded expansion. FOC($\{>\}$) is a fragment of the logic FOC($\mathbf{P}$), introduced by Kuske and Schweikardt in order to generalize first-order logic to counting problems [20]. FOC($\mathbf{P}$) is a very expressive counting logic and allows counting quantifiers $\#\bar{y}\varphi(\bar{x},\bar{y})$, which count for how many $\bar{y}$ the FOC($\mathbf{P}$)-formula $\varphi(\bar{x},\bar{y})$ is true. Moreover, arithmetic operations are allowed as well as all predicates in $\mathbf{P}$, which might contain comparisons, equivalence modula a number, etc. Kuske and Schweikardt showed that the FOC($\mathbf{P}$)-model checking problem is fixed parameter tracktable on graphs of bounded degree and hard on trees of bounded depth. The fragment FOC($\{>\}$) is more restrictive and allows only counting quantifiers of single variables and no arithmetic operations. The only predicate is comparison against an arbitrary number, but not between counting terms. While FOC($\{>\}$)-model checking is still hard on trees of bounded depth, there is an "approximation scheme" for FOC($\{>\}$) on classes of bounded expansion [8]: An algorithm gives either the right answer or says "mayby," but only if the formula is both almost satisfied and not satisfied. For a fragment of FOC($\{>\}$), which captures in particular the partial dominating set problem, we can compute even an exact answer to the model checking problem in linear fpt time [8]. That fragment consists of formulas of the form

$$\exists x_1 \dots \exists x_k \# y\, \varphi(y, x_1, \dots, x_k) > N, \tag{1}$$

where $\varphi$ is a first-order formula and $N$ an arbitrary number. The semantics of the *counting quantifier* $\#y\,\varphi(y, v_1, \dots, v_k)$ is the number of vertices $u$ in $G$ such that $G$ satisfies $\varphi(u, v_1, \dots, v_k)$. As an example, the existence of partial dominating set can be expressed as

$$\exists x_1 \dots \exists x_k \# y \bigvee_{i=1}^{k} E(y, x_i) \vee y = x_i > t, \tag{2}$$

where $k$ is the number of the dominating, and $t$ the number of dominated vertices. The length of the formula only depends on $k$. This implies that partial dominating set can be solved in linear fpt time on classes of bounded expansion.

There is another fragment of FOC($\mathbf{P}$), which should not be confused with FOC($\{>\}$). In FOC$_1$($\mathbf{P}$), introduced by Grohe and Schweikardt [16], the counting terms may contain at most one free variable. They show that FOC$_1$($\mathbf{P}$) is fixed-parameter tractable on nowhere dense graph classes [16]. Note that formula 2 is in FOC($\{>\}$) but not in FOC$_1$($\mathbf{P}$) as the counting term relies on $k$ free variables. Hence, FOC($\{>\}$) and FOC$_1$($\mathbf{P}$) are orthogonal in there expressiveness.

**Table 1** Results of this paper (in boldface) and some related known results. *Hard* means at least W[1]-hard. *PDS like* indicates problems similar to the partial dominating set problems: All problems that can be expressed by a FOC({>}) formula of the form (1). The mentioned approximation results are quite different. Numbers are approximated either with a relative or an absolute error.

| Graph class | FO-MC | $\mathrm{FOC}_1(\mathbf{P})$ | FOC({>}) | PDS like |
|---|---|---|---|---|
| bounded expansion | fpt [10] | fpt [16] | hard [8]<br>$(1+\varepsilon)$-approx fpt [8] | fpt [8] |
| nowhere dense | fpt [15] | fpt [16] | hard, approx open | **fpt**[c] |
| almost nowhere dense | **hard**[a] | open | **hard**[a] | **hard**[a]<br>**approx**$\pm\delta$ **fpt**[b] |
| general graphs | hard | hard | hard | hard |

[a] Corollary 22, [b] Corollary 2, [c] Theorem 1

There has been some research about *low degree graphs*. A graph class has low degree if every (sufficiently large) graph has degree at most $n^\varepsilon$ for every $\varepsilon > 0$. Examples are classes with bounded degree or classes with degree bounded by a polylogarithmic function. These graph classes are incomparable to nowhere dense classes. Especially, classes of low degree are not closed under subgraphs. On those classes, Grohe has shown that first-order model-checking can be solved in almost linear time [14]. Recently, Durand, Schweikardt, and Segoufin have generalized the result to query counting with constant delay and almost linear preprocessing time [9]. Vigny explores dynamic query evaluation on graph classes with low degree [24].

*Almost nowhere dense* is a property which subsumes both low degree and nowhere dense classes. Whereas a nowhere dense class $\mathcal{C}$ can be characterized that for every $r$ graphs do not contain up to $r$ times subdivided cliques of arbitrary sizes, for an almost nowhere dense class arbitrary sizes are allowed, but their growth must be bounded by subpolynomial function in the size of the graph.

Due to space limitations in this extended abstract many proofs, definitions, results, and comments can be found only in the appendix, which contains a full version of this paper. Of course, all main results are presented in this short version as well.

## 1.1 Our Results

In this work, we consider a fragment of FOC({>}), which we will call *PDS-like formulas*, namely formulas of the form

$$\exists x_1 \dots \exists x_k \# y \, \varphi(y, x_1, \dots, x_k) > N$$

for a quantifier-free FO-formula $\varphi$ and an (arbitrarily big) number $N \in \mathbf{Z}$. This logic is strong enough to express the partial dominating set problem as formula (2) is contained in the fragment described above. Remember that this fragment and $\mathrm{FOC}_1(\mathbf{P})$ are orthogonal. Table 1 contains an overview of most of the results in this paper.

In formulas that start with existential quantifier it is natural to ask for a witness, if we can indeed fulfill the formula. For example, in the partial dominating set problem we are usually interested in actually finding the dominating set rather than verify than one exists. Often, this is not an issue as problems are self-reducible. Using self-reducibility to find a witness incurs a runtime penalty. The next theorem shows that solving the model checking problem, and finding a witness, for formulas in the form of 1 is possible.

▶ **Theorem 1.** *Let $C$ be a nowhere dense graph class. For every $\varepsilon > 0$, every graph $G \in \mathcal{C}$ and every quantifier-free first-order formula $\varphi(y\bar{x})$ we can compute a vertex tuple $\bar{u}^*$ that maximizes $[\![\#y\,\varphi(y\bar{u}^*)]\!]^G$ in time $O(n^{1+\varepsilon})$.*

As an immediate corollary, we get that the model-checking problem for PDS-like formulas and thus, also the partial dominating set problem are solvable in almost linear fpt time on nowhere dense graph classes, where the parameter is the length of the formulas or the solution size $k$ respectively. Moreover, our meta-algorithm does not only work for partial dominating set, but for variants such as partial total or partial connected dominating set as well.

Note that Theorem 1 does not follow from the fact that model-checking for $\mathrm{FOC}_1(\mathbf{P})$ or that query-counting for FO-logic is fixed-parameter tractable [16] as we do not count the number of solutions to a query, but the number of witnesses to some solution. Also, PDS-like formulas form a fragment orthogonal to $\mathrm{FOC}_1(\mathbf{P})$. Moreover, we were not able to prove Theorem 1 by using the result from [16] as a subroutine: formulas inside a counting quantifier are allowed to have at most one free variable and this weakens self-reducibility or similar techniques drastically.

The above theorem cannot be extended to the more general case of almost nowhere dense graph classes. It turns out that even for non-counting formulas this is not possible, as the (classical) dominating set problem becomes W[1]-hard on some almost nowhere dense graph classes. This lower bound implies as a special case that plain FO-model checking is intractable on some almost nowhere dense graph classes. As far as we are aware this does not follow directly from previously known results.

However, we can go beyond nowhere dense classes if we do not insist on an exact solution: The model-checking problem for PDS-like formulas can be approximated with an *additive* subpolynomial error in almost linear fpt time on almost nowhere dense classes of graphs. To be more precise, we get the following, slightly more general result.

▶ **Corollary 2.** *Let $\mathcal{C}$ be an almost nowhere dense class of graphs. For every $\varepsilon > 0$, every graph $G \in \mathcal{C}$ and every quantifier-free first-order formula $\varphi(y\bar{x})$, we can compute in time $O(n^{1+\varepsilon})$ a vertex tuple $\bar{u} \in V(G)^{|\bar{x}|}$ with*

$$|\max_{\bar{u}} [\![\#y\,\varphi(y\bar{u})]\!]^G - [\![\#y\,\varphi(y\bar{u}^*)]\!]^G| \leq n^{\varepsilon}.$$

Talking about characterizations of almost nowhere dense graph classes, we provide a plethora of different characterizations, similar to the ones for bounded expansion and nowhere denseness. We show that a class is almost nowhere dense classes if and only if measures like $r$-shallow (topological) minor, forbidden $r$-subdivisions and (weak) $r$-coloring numbers are bounded by $f(r,\varepsilon)n^{\varepsilon}$.

We also examine almost nowhere dense classes from an algorithmic point of view: Whereas it is "natural" to consider monotonicity as closure property for nowhere dense graph classes, it is similarly natural to consider closure under edge deletion for almost nowhere dense graph classes. Consider a graph class $\mathcal{C}$ which is closed under deleting edges. Then we show that the problem of finding an $r$ times subdivided $k$-clique is fpt for every fixed $r$ on $\mathcal{C}$ if and only if $\mathcal{C}$ is almost nowhere dense. In particular, for every graph class that is not almost nowhere dense, but closed under deletion of edges, there exists a number $r$ such that finding $r$-subdivided $k$-cliques cannot be solved in fpt time under some complexity theoretic assumption, and, therefore, the FO model checking problem for formulas of the form $\exists\bar{x}\varphi(\bar{x})$ where $\varphi(\bar{x})$ is quantifier free and has predicates for adjacency and distance-$r$ adjacency, cannot be solved either. The situation for distance-$r$ independent set is different: Like finding an $r$-times subdivided clique it is fpt on almost nowhere dense graph classes, but there exists a graph class which is not almost nowhere dense and is closed under edge deletion where the problem is fpt.

## 1.2 Techniques

For Theorem 1, we use a novel dynamic programming technique on game trees of Splitter games. Splitter games were introduced by Grohe, Kreutzer, and Siebertz [15] to solve the first-order model-checking problem on nowhere dense classes. Together with their new concept of sparse neighborhood covers they achieved small recursion trees of constant depth.

Splitter games can be understood as a localized variation of the cops and robbers game for bounded treedepth (not to be confused with locally bounded treedepth). In contrast to [15] we apply a dynamic programming approach, similar to the ones used on bounded tree-depth decompositions. In contrast to bounded treedepth, a graph decomposes into neighborhoods of small radius instead of connected components when removing vertices according to Splitter's winning strategy. A challenge is that the resulting neighborhoods – in contrast to connected components – are not disjoint and lead to double counting for counting problems (an issue that does not occur in FO-model checking). To avoid double counting we introduce so-called cover systems specifically for the subgraph "induced" by the solution. The existence of such cover systems shows that there is a disjoint selection of small neighborhoods that cover all the vertices relevant to our counting problem. By solving a certain variation of the independent set problem, we can find such a selection and can safely combine the results of local parts of the graph as in dynamic programs for bounded tree-depth.

To achieve our second result Corollary 2, we adapt the techniques of the proof for solving the corresponding exact counting problem on classes of bounded expansion [8]: We replace $\#y\,\varphi(y\bar{x})$ by a sum of gradually simpler counting terms until they are simple enough to be easily evaluated. During this process we use transitive fraternal augmentations and a functional representation to encode necessary information into the graph, which is needed during the above simplification of counting terms. Along the way some difficult to handle literals appear in only a few number of terms. Ignoring them leads to the imprecision of our approximation. As the number of functional symbols in (almost) nowhere dense graph classes is not bounded by a constant as it is the case in classes of bounded expansion, the techniques from [8] have to adapted and extended. The main problem why their proof cannot be used directly is that the replacement of formulas leads to formulas of constant size in the case of bounded expansion, but to a non-constant size in our case. Here we use some new tricks and observe, that even though the transformed formulas can be of subpolynomial length, they can basically be replaced by many short formulas.

# 2 Preliminaries

## 2.1 Weak coloring numbers

A central concept in this paper are generalized coloring numbers, especially the weak coloring numbers introduced by Kierstead and Yang [17]. An *ordering* $\pi$ of a graph $G$ is a linear ordering of its vertex set and the set of all such orderings is denoted by $\Pi(G)$.

▶ **Definition 3** (Kierstead and Yang [17]). *A vertex $u \in V$ is* weakly $r$-reachable *from a vertex $v \in V$ with respect to $\pi \in \Pi(G)$ if $u \leq_\pi v$ and there exists a path $P$ from $u$ to $v$ of length at most $r$ such that $u \leq_\pi w$ for each $w \in V(P)$. The set of weakly $r$-reachable vertices from $v$ with respect to $\pi$ is denoted by $\mathrm{WReach}_r[G, \pi, v]$. Note that $v$ is always included in this set. We write $\mathrm{wdist}_{G,\pi}(u, v) \leq d$ if $u \in \mathrm{WReach}_r[G, \pi, v]$ or $v \in \mathrm{WReach}_r[G, \pi, u]$.*

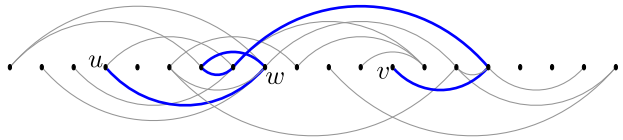



**Figure 1** $u$ is weakly 5-reachable from $v$ by the highlighted path, but $w$ is not weakly reachable from $v$.

*The* weak $r$-coloring number *of a graph $G$ (and an ordering $\pi$) is defined as*

$$\mathrm{wcol}_r(G,\pi) := \max_{v\in V(G)} |\mathrm{WReach}_r[G,\pi,v]|$$

$$\mathrm{wcol}_r(G) := \min_{\pi\in\Pi(G(V))} \mathrm{wcol}_r(G,\pi).$$

The weak 1-coloring number of a graph is one more than its degeneracy, which is the smallest number $d$ such that every subgraph $H \subseteq G$ has a vertex of degree at most $d$ in $H$. The weak coloring number can be seen as a localized version of tree-depth, as

$$\mathrm{wcol}_1(G) \leq \mathrm{wcol}_2(G) \leq \cdots \leq \mathrm{wcol}_\infty(G) = \mathrm{td}(G) \text{ [21].}$$

Figure 1 contains an example of weak $r$-reachability. Weak coloring numbers can be used to characterize nowhere dense graph classes:

▶ **Proposition 4** ([25, 22]). *A graph class $\mathcal{C}$ is nowhere dense if and only if there exists a function $f$ such that for every $r \in \mathbf{N}$, every $\varepsilon > 0$, every graph $G \in \mathcal{C}$ satisfies* $\mathrm{wcol}_r(H) \leq f(r,\varepsilon)|H|^\varepsilon$ *for every $H \subseteq G$.*

## 2.2 Splitter game

We will use a game-based characterization of nowhere denseness introduced by Grohe, Kreutzer and Siebertz [15]. Given a graph $G$, a radius $r$ and a number of rounds $\ell$, *the $(\ell,r)$-Splitter game* on $G$ is an alternating game between two players called *Splitter* and *Connector*. The game starts on $G_0 = G$. In the $i$th round, the Connector chooses a vertex $v_i$ from $G_i$. Then the Splitter chooses a vertex $s_i$ from the radius-$r$ neighborhood of $v_i$ in $G_i$. The game continues on $G_{i+1} = G_i[v_i] - s_i$. Splitter wins if after $\ell$ rounds the graph is empty. Grohe, Kreutzer and Siebertz showed that nowhere dense graph classes can be characterized by Splitter games:

▶ **Proposition 5** ([15]). *Let $\mathcal{C}$ be a nowhere dense class of graphs. Then, for every $r > 0$, there is $\ell > 0$, such that for every $G \in \mathcal{C}$, Splitter has a strategy to win the $(\ell,r)$-splitter game on $G$.*

Note that a winning move of Splitter in a current play can be computed in almost linear time [15, Remark 4.7].

## 2.3 Sparse neighborhood covers

Even though the splitter game ends after a bounded number of rounds $\ell$ for nowhere dense classes, the game tree, i.e. the tree spanned by all possible plays of Splitter and Connector, can still be large, e.g. in the dimensions of $n^\ell$. To make the game trees small and useful for algorithmic use, Grohe, Kreutzer and Siebertz introduced sparse neighborhood covers [15]. These covers group "similar" neighborhoods into a small number cluster of bounded radius. These clusters can be used instead of the neighborhoods, reducing the size of the game tree to $O(n^{1+\varepsilon})$.

▶ **Definition 6** ([15]). *For a radius $r \in \mathbf{N}$, an* $r$-neighborhood cover *$\mathcal{X}$ of a graph $G$ is a set of connected subgraphs of $G$ called* clusters*, such that for every vertex $v \in V(G)$ there is some $X \in \mathcal{X}$ with $N_r[v] \subseteq V(X)$. The degree of $v$ in $\mathcal{X}$ is the number of clusters that contain $v$ and the radius of $\mathcal{X}$ is the maximal radius of a cover in $\mathcal{X}$. A class $\mathcal{C}$ admits* sparse neighborhood covers *if there exists $c \in \mathbf{N}$ and for all $r \in \mathbf{N}$ and all $\varepsilon > 0$ a number $d = d(r, \varepsilon)$ such that every graph $G \in \mathcal{C}$ admits an $r$-neighborhood cover of radius at most $c$ and degree at most $d|G|^{\varepsilon}$.*

▶ **Proposition 7** ([15]). *Every nowhere dense class $\mathcal{C}$ of graphs admits a sparse neighborhood cover. For a graph $G \in \mathcal{C}$ and $r \in \mathbf{N}$ such an $r$-neighborhood cover can be computed in time $f(r, \varepsilon)n^{1+\varepsilon}$ for every $\varepsilon > 0$.*

Indeed, the existence of such covers is another characterization of nowhere dense classes.

▶ **Definition 8.** *For a graph $G$ with a vertex order $\pi$, $r \in \mathbf{N}$ and a vertex $v \in V(G)$, we define $X_r[G, \pi, v]$ as $\{u \in V(G) \mid v \in \mathrm{WReach}_r[G, \pi, u]\}$. We let $\mathcal{X}_r = \{X_{2r}[G, \pi, v] \mid v \in V(G)\}$.*

From the proof of Proposition 7 it follows, that the set family $\mathcal{X}_r$ is such a sparse neighborhood cover.

## 2.4 Low treedepth colorings

A crucial algorithmic tool in the study of bounded expansion and nowhere dense graph classes are *low treedepth colorings*, also known as $r$-centered colorings.

▶ **Definition 9.** *An $r$-treedepth coloring of a graph $G$ is a coloring of vertices of $G$ such that any $r' \leq r$ color classes induce a subgraph with treedepth at most $r'$.*

The following statement by Zhu [25] is modified such that it is constructive and holds also for a given vertex ordering $\pi$. It follows from the original proof.

▶ **Proposition 10** ([25, Proof of Thm. 2.6]). *If $\pi$ is a vertex ordering of a graph $G$ with $\mathrm{wcol}_{2^{r-2}}(G, \pi) \leq m$, an $r$-treedepth coloring can be computed with at most $m$ colors in time $O(mn)$.*

Graph classes of bounded expansion can be characterized by low treedepth colorings, i.e., each graph has an $r$-treedepth coloring with at most $f(r)$ many colors.

# 3 Exact Evaluation on Nowhere Dense Classes

In this section we consider the model-checking problem for formulas $\exists x_1 \dots x_k \# y\, \varphi(y\bar{x}) > N$ on nowhere dense graph classes for quantifier-free first-order formulas $\varphi$. We show that this problem can be solved in almost linear fpt time by solving its optimization variant $\max_{\bar{u} \in V(G)^{\bar{x}}} \# y\, [\![\varphi(y\bar{u})]\!]$.

## 3.1 Radius-$r$ Decomposition Tree

In the following, we will introduce a new kind of decomposition, which heavily relies on the ideas from [15]. We call it the radius-$r$ decomposition tree. For illustration, consider a tree-depth decomposition of a graph $G$. It has the property that after the removal of the root $v$ in the decomposition, for each connected component $C$ of $G - v$ there exists a child of $v$ in the decomposition that contains $C$. In the radius-$r$ decomposition tree, not every connected component is represented by a child but every radius-$r$ neighborhood of $G - v$

instead. Another difference is that these neighborhoods are not necessarily disjoint. We will use this radius-$r$ decomposition tree as the structure on which a dynamic program will solve $\max_{\bar{u}} [\![\#y\, \varphi(y\bar{u})]\!]^G$.

▶ **Definition 11.** *Let $G$ be a graph. Let $r, \ell \in \mathbf{N}$ be such that splitter has a winning strategy for the $\ell$-round radius-$2r$ splitter game on $G$. Let $\pi$ be an ordering of $G$.*

*A* radius-$r$ decomposition tree $T_r(G, \pi, \ell)$ *is a pair $(T, \beta)$ where $T$ is a tree of depth $\ell$ and $\beta\colon V(T) \to V(G)$. We construct it recursively. If $G$ is empty, $T_r(G, \pi, \ell)$ is the empty tree.*

*Let $s \in V(G)$ be the first move of the winning strategy of splitter for the $(\ell, 2r)$-splitter game on $G$. The root is a node $t$ with $\beta(t) = s$. For every $v \in V(G)$ we append the decomposition tree $T_r(G[X_v], \pi, \ell - 1)$ where $X_v = X_{2r}[G - s, \pi, v]$.*

Note that the case $\ell = 0$ while the graph is not empty, cannot happen due to the Splitter having a winning strategy.

▶ **Corollary 12.** *Let $G$ be a graph, $\pi$ a vertex ordering of $G$, $r, \ell \in \mathbf{N}$ and $T = T_r(G, \pi, \ell)$ a radius-$r$ decomposition tree. Let $t \in V(T)$ be a node and $T_t$ be the subtree of $T$ starting at $t$. Then for every $u \in W := \beta(V(T_t)) \setminus \{\beta(t)\}$ there exists a child $t'$ of $t$ such that $N_r^{G[W]}[u] \subseteq \beta(T_{t'})$.*

▶ **Lemma 13.** *Let $G$ be a graph, $\pi$ a vertex ordering of $G$ and $r, \ell \in \mathbf{N}$. Then, the radius-$r$ decomposition tree $T = T_r(G, \pi, \ell)$ (Definition 11) has size $|T| \leq \mathrm{wcol}_{2r}(G, \pi)^\ell n$ and depth $\ell$. The construction time is linear in $|T|$.*

**Proof.** By construction, the depth of the tree is determined by the depth of the splitter game, which is $\ell$.

Consider the root path $P_t$ of some node $t \in V(T)$. Then $\beta(P_t) \subseteq \mathrm{WReach}_{2r}[G, \pi, \beta(t)]$. As the length of $P_t$ is at most $\ell$, $\beta(t)$ appears at most $\mathrm{WReach}_{2r}[G, \pi, \beta(t)]^\ell \leq \mathrm{wcol}_{2r}(G, \pi)^\ell$ times (as a $\beta$-label of nodes) in $T$. Thus, $|T| \leq \mathrm{wcol}_{2r}(G, \pi)^\ell n$. ◀

▶ **Corollary 14.** *Let $\mathcal{C}$ be a nowhere dense graph class. For every $r \in \mathbf{N}$ the $r$-decomposition tree has constant depth, almost linear size and can be computed in almost linear time.*

## 3.2 Cover Systems

Given a subgraph $H$ in $G$ with a vertex ordering $\pi$ of $G$. A *cover system* of $H$ in $G$ is a family $\mathcal{Z}$ of clusters $Z_i = X_r[G, \pi, v] \in Z$ for some $r \in \mathbf{N}$ such that every connected component $C$ of $H$ is contained in some $Z_i$. A cover system is *non-overlapping* if all distinct clusters have an empty intersection.

▶ **Lemma 15.** *For every graph $G$ with a vertex ordering $\pi$, every $D \subseteq V(G)$ of size $k$, there exists a cover system of $G[N[D]]$ in $G$ of size at most $k$ where each cluster has the same radius $r \leq 2^k$.*

**Proof.** We start with the clusters $X_2[G, \pi, \min_\pi N[d]]$ for every $d \in D$. Call this collection $\mathcal{Z}$. Note that $\mathcal{Z}$ is already a valid cover system of $G[N[D]]$ in $G$. If two distinct clusters $X_r[G, \pi, z]$ and $X_r[G, \pi, z']$ from $\mathcal{Z}$ intersect, we replace both with a new cluster $X_{2r}[G, \pi, \min_\pi\{z, z'\}]$ in $\mathcal{Z}$. Every vertex or edge covered by the two old clusters stays covered in the new one. Also, if two clusters $X_r[G, \pi, z]$ and $X_{r'}[G, \pi, z']$ are of a different radius, say, $r' < r$, we replace $X_{r'}[G, \pi, z']$ with $X_r[G, \pi, z']$ to match the radii of all the clusters.

We repeat this until no intersecting clusters remain. As the number of clusters decreases with every step, the radius is at most $2^k$ at the end. ◀

For Theorem 1, one needs to find clusters from $\mathcal{X}_r$ which are disjoint and maximize the sum of weights of clusters. This is captured by the following definition. We can solve this problem in almost linear time on nowhere dense graph classes, by noticing that the intersection graphs of the sparse neighborhood covers $\mathcal{X}_r$ are almost nowhere dense. Then, one can use treedepth colorings and LinEMSOL.

▶ **Definition 16** (Disjoint Cluster Maximization). *Given a graph, a set system $\mathcal{X}_r$ as defined in Definition 8, labelled by a function $\Lambda : \mathcal{X}_r \to 2^\Lambda$ of size $k$. Each combination of a cluster $X \in \mathcal{X}_r$ and label $\lambda \in \Lambda(X)$ is weighted by a function $w$.*

*Problem: Find pairwise disjoint clusters $X_1, \dots, X_k \in \mathcal{X}_r$ such that for each label $\lambda_i \in \Lambda$ the cluster $X_i$ is labeled $\lambda_i$ and $X_1, \dots, X_k$ maximize $\sum_{i=1}^{k} w(X_i, \lambda_i)$ for such cluster sets.*

*Parameter: $r, k$*

Let $\Omega$ be the set of weighted positive conjunctive clauses $(\mu, \omega(y\bar{x}))$, $\bar{z} \subseteq \bar{x}$ and $\bar{u} \in V(G)^{|\bar{x}|}$. With $\Omega|_{\bar{z}}$ we denote a subset of $\Omega$ with weighted clauses $(\mu, \omega(y\bar{z}))$ where every variable occurring in $\omega$ is from $\bar{z}$. We define $\Omega|_{\bar{z}}[Z, \bar{u}]$ as $\sum_{v \in Z} \sum_{(\mu,\omega) \in \Omega|_{\bar{z}}} \mu[\![\omega(v\bar{u})]\!]^G$. Note that $\Omega|_{\bar{z}}[Z, \bar{u}]$ depends only on the assignment of $\bar{z}$ and does not need the full assignment $\bar{u}$ of $\bar{x}$.

To illustrate the following lemma, consider a positive conjunctive clause $\omega(y\bar{x}\bar{z})$, sets $P, W \subseteq V(G)$ and $\bar{u} \in P^{\bar{x}}, \bar{w} \in W^{\bar{z}}$. To count the fulfilling vertices $v \in W$ of $\omega$, i.e. $\Omega[W, \bar{u}]$, we want to reduce this task to counting on cover systems of $N[\bar{w}]$. However, as not all fulfilling vertices in $W$ are adjacent to $\bar{w}$, we need to be more careful.

▶ **Lemma 17.** *Let $G$ be a graph, $\Omega$ a set of weighted positive conjunctive clauses $(\mu, \omega(y\bar{x}\bar{z}))$, $P, W \subseteq V(G)$ disjoint, $\bar{u} \in P^{\bar{x}}, \bar{w} \in W^{\bar{z}}$ such that $N[\bar{w}] \subseteq P \cup W$. For every cover system $\mathcal{Z}$ of $G[N[\bar{w}]]$ in $G[W]$ it holds that*

$$\Omega[W, \bar{u}\bar{w}] = \Omega|_{y\bar{x}}[W, \bar{u}\bar{w}] + \sum_{Z \in \mathcal{Z}} (\Omega|_{y\bar{x}\bar{z}_Z}[Z, \bar{u}\bar{w}] - \Omega|_{y\bar{x}}[Z, \bar{u}])$$

*where $\bar{z}_Z$ are the variables $z_i$ from $\bar{z}$ which are assigned to a vertex in $Z$.*

Let us consider how a solution $\bar{u}$ for $\#y\,\varphi(y\bar{x})$ interacts with a radius-$r$ decomposition of the input graph $G$ where $r$ is chosen appropriately big, e.g. $2^k$ resulting from Lemma 15. First, we transform $\varphi$ into a set of positive clauses $\Omega$, making the application of Lemma 17 possible.

Consider some node $t$ in $T_r$. When applying Lemma 17 with $P$ as the vertices of the root path of $t$ and $W$ as $T_t$, we see that the resulting cover system $\mathcal{Z}$ corresponds to a selection of children of $t$ in $T_r$, as both use the sets $X_r$ from Definition 8. Now imagine that we know $\Omega_{y\bar{x}\bar{z}_Z}[Z, \bar{u}]$ for every $Z \in \mathcal{Z}$. Note that this number only depends on the assignment of $\bar{x}\bar{z}_z$ and not the vertices assigned outside $P$ and $Z$. With Lemma 17 we can combine these numbers into $\Omega[W, \bar{u}]$ without needing to know the actual assignments of $\bar{z}_Z$ in the cover system anymore! Note that $\Omega_{y\bar{x}}[Z]$ is easily computable while only knowing $\bar{u}$ and not $\bar{w}$.

Thus, we can compute $[\![\#y\,\varphi(y\bar{u})]\!]$ bottom-up using the radius-$r$ decomposition while only considering the vertices assigned in $\bar{u}$ which are contained in the root path of the considered vertex.

## 3.3 Dynamic Program

To determine $\max_{\bar{u}} \#y\,\varphi(y\bar{u})$ for a quantifier-free formula $\varphi(y\bar{x})$ we recursively compute the following information in the decomposition tree of $G$ (bottom-up, if you will). Consider some node $t$ of $T$ and a partial assignment $\alpha$ of $\bar{x}$ to the root path $\beta(P_t)$. The interesting

information is: How many vertices underneath $t$, i.e. in $V(G_t)$, fulfill $\varphi$ under the "best" choice on completing the assignment $\alpha$ to vertices in $V(G_t)$. Then the answer to the problem can be read off the information for the root node.

Assume we already know this kind of information for every child $t'$ of $t$. To compute this information for $t$, we branch how the variables $x_i$ that are not assigned under $\alpha$ are distributed among the children of $t$. Then the table entries of these children are combined in a suitable way. We do this for every distribution among children and take the maximum of the resulting values. If a vertex corresponding to $t$ fulfills with the assignment the formula $\varphi$, it gets counted towards the number of "fulfilling" vertices.

However, we have to take more into consideration. First, branching on the distribution of the unassigned variables $x_i$s under $\alpha$ among the children of $t$ is not fast enough, as there are around $n^k$ possibilities for that. Instead, we branch on how the unassigned variables are partitioned. For every such partition, we formalize the optimal choice of children $t_i$ such that they contain exactly the unassigned variables from the $i$-th part, as an optimization problem.

Secondly, the graphs $G_{t'}$ spanned by each child $t'$ of $t$ are in general not disjoint. Combining the counts of two overlapping graphs yields to double counting. We circumvent this in the above optimization problem.

Thirdly, we need to keep track of how the vertices in the root path $P_t$ are adjacent to the variables $x_i$ that are assigned underneath $t$. We cannot branch on the complete assignment as the number of those is too high.

Before we turn to the dynamic program on the decomposition tree, we consider something simpler:

Let $G$ be a graph and $\varphi(y\bar{x})$ be quantifier-free FO formula. Consider the pair $(P, W)$ which is a set of vertices $P = \{v_1, \dots, v_k\} \subseteq V(G)$ and a set $W \subseteq V(G)$ that is disjoint with $P$. We are interested in how many vertices $v$ in $G[P \cup W]$ satisfy $\varphi(v\bar{u})$ for an optimal choice of $\bar{u} \in (P \cup W)^{|\bar{u}|}$. For this, we keep track of $M_\alpha^{(P,W)}[S]$, which is the number of fulfilling vertices $v \in W$ wrt. $\varphi$, $\hat{\alpha}$ and $S$, maximizing over $S$-completions $\hat{\alpha}$ on $W$.

We can "forget" a vertex $v$, i.e., derive the information of $(P, W \cup \{v\})$ from the information $(P \cup \{v\}, W)$ as follows: Assume the maximum number of fulfilling vertices in $W$ is $x$ for a given partial assignment $\alpha$ on $P \cup \{v\}$ and adjacency profile $S$ on $P \cup \{v\}$. Then the number of fulfilling vertices in $W \cup \{v\}$ is $x + 1$ if $v$ satisfies $\varphi$ with the assignment $\alpha$ and adjacency profile $S$, or $x$ otherwise. However, neither $\alpha$ nor $S$ are valid assignments or adjacency profiles for $P$. Hence, we need to adjust these so that we can formulate this information for $(P, W \cup \{v\})$. For this, we need to remove $v$ from $\alpha$ and add the neighborhood of $v$ in $P$ to $S$ as $S_i$, for every $i$ with $\alpha(x_i) = v$. Then, $M_\alpha^{(P\cup\{v\},W)}[S] = M_{\alpha|_P}^{(P,W\cup\{v\})}[S'](+1)$ where $\alpha|_P$ is the assignment $\alpha$ without $v$ and $S'$ is the adjacency profile as described above.

One can also combine the information of two structures $(P, W_1)$ and $(P, W_2)$ to get the information of $(P, W_1 \uplus W_2)$ if $W_1$ and $W_2$ are disjoint. This is also known as "merge." Consider some assignment $\alpha$ on $P$ and some adjacency profile $S$ on $P$. Then the number of fulfilling vertices in $U \uplus W$ wrt $\varphi$, $\alpha$ and $S$ is the $\max\{M_\alpha^{P,W_1}[S_1] + M_\alpha^{P,W_2}[S_2] \mid S_1 \uplus S_2 = S\}$.

Indeed however, the algorithm does not take a quantifier-free formula $\varphi$ but a set of weighted positive conjunctive clauses. Instead of just counting the fulfilled vertices, it computes the added up weight of them wrt. to the weights of the clauses.

▶ **Theorem 1.** *Let $C$ be a nowhere dense graph class. For every $\varepsilon > 0$, every graph $G \in \mathcal{C}$ and every quantifier-free first-order formula $\varphi(y\bar{x})$ we can compute a vertex tuple $\bar{u}^*$ that maximizes $[\![\#y\, \varphi(y\bar{u}^*)]\!]^G$ in time $O(n^{1+\varepsilon})$.*

## 4 Characterizing Almost Nowhere Dense Graph Classes

In this section, we provide various characterizations of almost nowhere dense classes, i.a. via bounded depth minors and generalized coloring numbers.

▶ **Definition 18** (Almost nowhere dense). *A graph class $\mathcal{C}$ is* almost nowhere dense *if for every $r \in \mathbf{N}$, $\varepsilon > 0$ there exists $n_0$ such that no graph $G \in \mathcal{C}$ with $|G| \geq n_0$ contains $K_{\lceil |G|^\varepsilon \rceil}$ as a depth-$r$ minor.*

▶ **Theorem 19.** *Let $\mathcal{C}$ be a graph class. The following statements are equivalent.*

1. *$\mathcal{C}$ is almost nowhere dense.*
2. *For every $r \in \mathbf{N}$, $\varepsilon > 0$ there exists $n_0$ such that no graph $G \in \mathcal{C}$ with $|G| \geq n_0$ contains $K_{\lceil |G|^\varepsilon \rceil}$ as a depth-$r$ minor.*
3. *For every $r \in \mathbf{N}$, $\varepsilon > 0$ there exists $n_0$ such that no graph $G \in \mathcal{C}$ with $|G| \geq n_0$ contains $K_{\lceil |G|^\varepsilon \rceil}$ as a depth-$r$ topological minor.*
4. *For every $r \in \mathbf{N}$, $\varepsilon > 0$ there exists $n_0$ such that no graph $G \in \mathcal{C}$ with $|G| \geq n_0$ contains an $r'$-subdivision of $K_{\lceil |G|^\varepsilon \rceil}$ as a subgraph for any $r' \leq r$.*
5. *For every $r \in \mathbf{N}$, $\varepsilon > 0$ there exists $n_0$ such that $\operatorname{wcol}_r(G) \leq |G|^\varepsilon$ for every graph $G \in \mathcal{C}$ with $|G| \geq n_0$.*
6. *For every $r \in \mathbf{N}$, $\varepsilon > 0$ there exists $n_0$ such that $\operatorname{col}_r(G) \leq |G|^\varepsilon$ for every graph $G \in \mathcal{C}$ with $|G| \geq n_0$.*

The characterizations from Theorem 19 are very similar to those for nowhere dense classes. The only difference in the characterizations 1. to 4. would be the size of the forbidden cliques: for nowhere dense classes, the size would be $f(r)$ instead of $\lceil |G|^\varepsilon \rceil$. Similarly, if we would substitute "for every $G \in \mathcal{C}$" with "for every subgraph $G \subseteq H \in \mathcal{C}$" in characterizations 5 and 6 would characterize nowhere dense classes. Note that every almost nowhere dense class which is monotone, i.e. closed under taking subgraphs, is also nowhere dense.

Conversely, if a class $\mathcal{C}$ is almost nowhere dense, then its subgraph-closure $\mathcal{C}_\subseteq$ is not almost nowhere dense in general. Consider for this the class of graphs which for every $n \in \mathbf{N}$ contains independent set of size $n$ with a clique of size $\log n$, i.e. the graph $I_n \cup K_{\log n}$. This class is almost nowhere dense but its subgraph-closure contains cliques $K_n$ of every size $n$ as member, and so, all graphs.

## 5 Approximation on Almost Nowhere Dense

In this section we consider the same problem as before, i.e., finding vertices for $\bar{x}$ that satisfy $\#y\,\varphi(\bar{x}y) > N$ but on almost nowhere dense classes of graphs. Here, we give an approximation algorithm with an additive error. For this, we use completely different techniques compared to Section 3. We first show how to reduce the corresponding model-checking problem to approximate sums over unary functions. Then we present the approximate optimization algorithm in Theorem 20.

The main result of this section is the following approximate optimization algorithm with additive error.

▶ **Theorem 20.** *There exists a computable function $f$ such that for every graph $G$ and every quantifier-free first-order formula $\varphi(y\bar{x})$ we can compute a vertex tuple $\bar{u}^*$ with*

$$|\max_{\bar{u}} [\![\#y\,\varphi(y\bar{u})]\!]^G - [\![\#y\,\varphi(y\bar{u}^*)]\!]^G| \leq 4^{|\varphi|} \operatorname{wcol}_2(G)^{O(|\varphi|)}$$

*in time $\operatorname{wcol}_{f(|\varphi|)}(G)^{f(|\varphi|)} n$.*

For the *approximate model-checking* problem with an additive error $\delta$, similar to [8], we want an algorithm such that

1. the algorithm returns "yes" only if $G$ satisfies the formula,
2. returns "no" only if $G$ does not satisfy the formula,
3. returns $\perp$ only if the optimum is within $\delta$ to $N$.

The option $\perp$ can be seen as "I do not know" as the computed result and the desired result are so close that the difference falls into the additive error $\delta$.

Given the approximate optimization algorithm from Theorem 20, we can easily build an approximate model-checking algorithm as described above for the formula $\exists \bar{x} \# y\, \varphi(y\bar{x}) > N$ by computing a vertex tuple $\bar{u}^*$ from the theorem. If $N - [\![\#y\, \varphi(y\bar{u}^*)]\!]^G \leq \delta$, answer $\perp$. Otherwise, answer "yes" or "no" according whether $[\![\#y\, \varphi(y\bar{u}^*)]\!]^G > N$ or not. Note that $\delta$ cannot be chosen freely as it depends on the graph (respectively, its weak coloring numbers).

The runtime of the algorithm from Theorem 20 is fpt if the weak $r$-coloring numbers are bounded by $n^\varepsilon$ for $r \leq f(|\varphi|)$. This is the case for almost nowhere dense classes. This is in contrast to the results of [8] where the running time of their algorithms is bounded by $f(\mathrm{wcol}_{f(|\varphi|)})||G||$ which is fpt on classes of bounded expansion but is not fpt on nowhere dense and almost nowhere dense classes.

This gives us the following corollary.

▶ **Corollary 2.** *Let $\mathcal{C}$ be an almost nowhere dense class of graphs. For every $\varepsilon > 0$, every graph $G \in \mathcal{C}$ and every quantifier-free first-order formula $\varphi(y\bar{x})$, we can compute in time $O(n^{1+\varepsilon})$ a vertex tuple $\bar{u} \in V(G)^{|\bar{x}|}$ with*

$$|\max_{\bar{u}} [\![\#y\, \varphi(y\bar{u})]\!]^G - [\![\#y\, \varphi(y\bar{u}^*)]\!]^G| \leq n^\varepsilon.$$

## 6 Hardness Results

In this section, we try to see how far the above result can or cannot be extended to either a bigger class of problems or to more general graph classes. Exemplary, we examine the distance-$r$ versions of the dominating set, independent set and clique problem. Note that in contrast to the section before, we do not consider the partial problem versions. We see that each of these problems behave differently in this context. The distance-$r$ dominating set problem is already hard for distance 1 on some almost nowhere dense graph classes, whereas distance-$r$ independent set and distance-$r$ clique are both fpt on almost nowhere dense graph classes.

As for graph classes, we consider classes that are closed under removing edges because monotone graph classes are very well understood and the notions of nowhere dense and almost nowhere dense coincide on those classes. Interestingly, for graph classes closed under removing edges the distance-$r$ clique problem is fpt for all distances $r$ if and only if the class is almost nowhere dense (under some complexity theoretic assumptions). However, there exist graph classes which are closed under removing edges but not almost nowhere dense that allow for fpt algorithms for the distance-$r$ independent set problem. The difference of behavior between distance-$r$ clique and distance-$r$ independent set is also intriguing as the FO-formulation of these problems has exactly one quantifier alternation for both.

### 6.1 Exact Evaluation Beyond Nowhere Dense Classes

The following lemma proves that dominating set is W[1]-hard on almost nowhere dense classes.

The class of bipartite graphs with sides $L$ and $R$ where $L$ has polylogarithmic size is almost nowhere dense: A witness for this is a vertex ordering that starts with $L$ and starts . Only the vertices from $L$ are weakly $r$-reachable from any vertex. Hence, $\mathrm{wcol}_r(G) \leq |L| + 1$ for each $r$.

▶ **Theorem 21.** *In bipartite graphs whose left side has $2k(k-1)\lceil\log(n)\rceil$ vertices and whose right side has $n$ vertices it is W[1]-hard to decide whether there are $\binom{k}{2}$ right-side vertices dominating all left-side vertices.*

**Proof.** We reduce from colorful clique. Assume we have a $k$-partite graph $G$ of size $n$ consisting of parts $V_0, \dots, V_{k-1}$ (each of a different color) and want to find a colorful clique of size $k$. Without loss of generality, we can assume $n$ to be large enough that $\binom{2\lceil\log(n)\rceil}{\lceil\log(n)\rceil-1} \geq n$. This means, we can find for each $v \in V(G)$ a unique binary encoding $\mathrm{enc}(v)$ of length $2\lceil\log(n)\rceil$ such that the first bit is set to one and in total exactly half the bits are set to one. Let $\overline{\mathrm{enc}}(v)$ be the binary complement of $\mathrm{enc}(v)$. We construct a bipartite graph $H$, whose left side is partitioned into cells $C_{ij}$ for $0 \leq i \neq j < k$, each of size $2\lceil\log(n)\rceil$, and whose right side will be specified soon. The vertices of each cell are ordered. When we say for a given vertex $v$ from the right side and cell $C$ that *$v$ is connected to $C$ according to a specified encoding*, we mean that for $1 \leq l \leq 2\lceil\log(n)\rceil$, $v$ is connected to the $l$th vertex of $C$ if and only if the $l$th bit in the encoding is set to one. For $0 \leq i < k$ we define

$$\mathrm{succ}_i(j) = \begin{cases} j+1 \bmod k & i \neq j+1 \bmod k \\ j+2 \bmod k & \text{otherwise.} \end{cases}$$

For all $0 \leq i < j < k$ and all $u \in V_i$ and $v \in V_j$ such that $uv \in E(G)$, add a vertex $x_{u,v}$ to the right side and

- connect $x_{u,v}$ to $C_{i,j}$ according to $\mathrm{enc}(u)$,
- connect $x_{u,v}$ to $C_{i,\mathrm{succ}_i(j)}$ according to $\overline{\mathrm{enc}}(u)$,
- connect $x_{u,v}$ to $C_{j,i}$ according to $\mathrm{enc}(v)$,
- connect $x_{u,v}$ to $C_{j,\mathrm{succ}_j(i)}$ according to $\overline{\mathrm{enc}}(v)$. ◀

We can reduce the aforementioned dominating set variation to the classical dominating set problem by connecting the right side to a fresh vertex.

▶ **Corollary 22.** *There exists an almost nowhere dense graph class $\mathcal{C}$ where the dominating set problem is $W[1]$-hard and cannot be solved in time $n^{o(k)}$ assuming ETH. This implies also the hardness of the fragments PDS-like, $\mathrm{FOC}_1(\mathbf{P})$, and $\mathrm{FOC}(\{>\})$ of $\mathrm{FOC}(\mathbf{P})$ on $\mathcal{C}$.*

Note that this result does not follow from the intractability result of FO-logic on subgraph-closed somewhere dense classes, i.e. not nowhere dense classes.

## 6.2 Beyond Distance One

We showed that the dominating set problem is $W[1]$-hard on some almost nowhere dense graph class. However, this is not true for the distance-$r$ clique and independent set problem.

Distance-$r$ clique and independent set on the other hand are fpt on almost nowhere dense graph classes. Here, we use low treedepth colorings to solve existential FO formulas. With the right formulation and inclusion-exclusion this works even for distance-$r$ independent set which cannot be expressed as a purely existential FO formula.

▶ **Theorem 23.** *There exists a computable function $f$ such that for every graph $G$ the distance-$r$ clique problem can be solved in time $\mathrm{wcol}_{f(k,r)}(G)^{f(k,r)}n$.*

**Proof.** We can solve this problem with the help of subgraph queries where each subgraph is an $\leq r$-subdivision of a $k$-clique. These subgraphs have less than $k^2(r+1)$ vertices and there are at most $(r+1)^{k^2}$ of them. Subgraph queries can be done by checking an existential FO-formula using Theorem 20. ◀

▶ **Theorem 24.** *There exists a computable function $f$ such that for each graph $G$ the distance-$r$ independent set problem can be solved in time* $\mathrm{wcol}_{f(k,r)}(G)^{f(k,r)}n$.

**Proof sketch.** We count specially designed subgraphs to solve this problem. These subgraphs encode that there are vertices $v_1, \dots v_k$ which have some distance $d(v_i, v_j)$ from each other. As the distance constraint "$d(v_i, v_j) \geq r+1$" for the distance-$r$ independent set problem cannot be expressed this way, we use inclusion-exclusion to compute the number of such graphs. To count them, we use low treedepth colorings whose number of colors are bounded by weak coloring numbers. ◀

▶ **Corollary 25.** *For every almost nowhere dense graph class $\mathcal{C}$, every $r \in \mathbf{N}$ and every real $\varepsilon > 0$ both the distance-$r$ clique problem and the distance-$r$ independent set problem can be solved in time $O(n^{1+\varepsilon})$ given a graph $G \in \mathcal{C}$.*

## 6.3 Beyond Almost Nowhere Dense

For graph classes that are closed under removing *vertices and edges*, i.e., monotone graph classes, we know a lot already. Most importantly, FO-model checking is fpt on such classes if and only if the class is nowhere dense (unless FPT = $W[1]$) [15]. We now want to consider graph classes that are only closed under removing *edges*. Here the concept of almost nowhere dense graph classes becomes interesting.

The following observation follows directly from characterization 6 in Theorem 19. If $\mathcal{P}$ is a parameterized problem that can be solved in time $\mathrm{col}_{f(k)}(G)^{f(k)}n$ and $\mathcal{C}$ is an almost nowhere dense graph class, then $\mathcal{P}$ can be solved on $\mathcal{C}$ in almost linear fpt time $f(k,\varepsilon)n^{1+\varepsilon}$ for every $\varepsilon > 0$. We complement this by showing that the distance-$r$ clique problem is most likely not fpt on all graph classes that are not almost nowhere dense, but closed under removing edges. Hence, under certain common complexity theoretic assumptions, if a graph class $\mathcal{C}$ is closed under removal of edges then distance-$r$ clique is fpt on $\mathcal{C}$ iff $\mathcal{C}$ is almost nowhere dense.

▶ **Theorem 26.** *Let $\mathcal{C}$ be a graph class that is not almost nowhere dense, but closed under removing edges. Then there exists a number $r$, such that one cannot solve the distance-$r'$ clique problem parameterized by solution size in fpt time on $\mathcal{C}$ for all $r' \leq r$ unless* i.o.W[1] $\subseteq$ FPT.

Similar hardness results in parameterized complexity are usually built on the hardness assumption FPT $\neq W[1]$. The complexity class *i.o.*W[1] should be read as "infinitely often in W[1]" and needs to be explained.

▶ **Definition 27.** *For a language $L$ and an integer $n$ let $L_n = L \cap \{0,1\}^n$. A language $L$ is in i.o.$C$ for a complexity class $C$ if there is some $L' \in C$ such that $L'_n = L_n$ for infinitely many input lengths $n$.*

Considering the infinite often variant *i.o.*$C$ of a complexity class $C$ is an established technique in complexity theory (i.e., [3, 2]). To prove our result, we show that a graph class $\mathcal{C}$ that is not almost nowhere dense, contains an infinite sequence of graphs having cliques of polynomial size as bounded depth topological minors. If $\mathcal{C}$ is also closed under removal of edges then

having bounded depth topological clique minors of size $n$ implies the existence of subdivisions of arbitrary graphs $H$ of size $n$ as induced subgraphs. Extra care needs to be taken to make sure that all paths connecting the principal vertices should be of equal length, since otherwise a reduction would need to try out an exponential number of possible length combinations to finally find the correct subdivision of $H$ that is contained in $\mathcal{C}$. The following corollary is a direct consequence of Theorem 19.4.

▶ **Corollary 28.** *Let $\mathcal{C}$ be some graph class that is not almost nowhere dense. Then there are $r$, $\varepsilon$ and an infinite sequence of strictly ascending numbers $n_0, n_1, \ldots$ such that for all $i \in \mathbf{N}$ there is a graph $G \in \mathcal{C}$ of order at most $n_i$ that contains an $r'$-subdivision of $K_{\lceil n_i^\varepsilon \rceil}$ as a subgraph for some $r' \leq r$.*

The consequence i.o.W[1] $\subseteq$ FTP is weaker than W[1] $\subseteq$ FPT. We could use the latter in Theorem 26 if we required a stronger precondition, i.e., that $\mathcal{C}$ has "witnesses" for input lengths $n_0, n_1, n_2, \ldots$ such that the gap between $n_i$ and $n_{i+1}$ is only polynomial. This approach has been used, e.g., in proving lower bounds on the running time of MSO-model checking in graph classes where the treewidth grows polylogarithmically [19, 12].

**Proof of Theorem 26.** Let $r$ and $\varepsilon$ be the constants (depending on $\mathcal{C}$) from Corollary 28. Assume that the distance-$(r+1)$ clique problem on $\mathcal{C}$ is fpt when parameterized by solution size. We will present a Turing reduction showing that the (usual) clique problem on the class of all graphs is infinitely often in FPT.

By Corollary 28 for infinitely many $n_0, n_1, \cdots \in \mathbf{N}$ there exists a graph from $\mathcal{C}$ of size at most $n_i^{1/\varepsilon}$ that contains an $r'$-subdivision of a clique of size $n_i$ as a subgraph for some $r' \leq r$. Let us pick one $n = n_i$. Suppose we want to decide whether a graph $G$ with $n$ vertices contains a clique of size $k$. Since $\mathcal{C}$ is closed under removal of edges, there exist $r' \leq r$, and $n \leq N \leq n^{1/\varepsilon}$ such that $\mathcal{C}$ contains a graph $H_{r',N}$ consisting of an $r'$-subdivision of $G$ together with $N$ isolated vertices. Now for all $k$, $G$ contains a clique of size $k$ iff $H_{r',N}$ contains a distance-$(r'+1)$ clique of size $k$. Assume for contradiction we had an algorithm that decides in time at most $f(r',k)n^c$ whether a graph in $\mathcal{C}$ of size $n$ contains an distance-$(r'+1)$ clique for $r' \leq r$. (For graphs not in $\mathcal{C}$, the algorithm may give a wrong answer, but we can modify it to construct and test a witness of a distance-$(r'+1)$ clique on yes-instances. Hence, we can assume that the algorithm never returns "no" on yes-instances.)

The existence of such an algorithm yields us an FPT algorithm for the $k$-clique problem on general graphs: For all $r' \leq r$, and $n \leq N \leq n^{1/\varepsilon}$, we run this (hypothetical) fpt algorithm in parallel on $H_{r',N}$ for $f(r',k)N^c$ time steps. Then $G$ contains a clique of size $k$ iff for at least one value of $r'$ and $N$ we have $H_{r',N} \in \mathcal{C}$ and $H_{r,N}$ contains a distance-$(r'+1)$ $k$-clique.

As the $k$-clique problem is $W[1]$-hard, we get the desired result. ◀

Note that this result does not extend to the distance-$r$ independent set problem. Consider the class of graphs where at least half of its vertices are isolated. Then the distance-$r$ independent set problem is trivially FPT for this graph class. However, this graph class is closed under removing edges, but it is not almost nowhere dense.

## References


1 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Implicit branching and parameterized partial cover problems. *J. Comput. Syst. Sci.*, 77(6):1159–1171, 2011. doi:10.1016/j.jcss.2010.12.002.

2 Richard Beigel, Lance Fortnow, and Frank Stephan. Infinitely-often autoreducible sets. *SIAM Journal on Computing*, 36(3):595–608, 2006.

3 Leonard Berman. On the structure of complete sets: Almost everywhere complexity and infinitely often speedup. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 76–80, 1976. doi:10.1109/SFCS.1976.22.

4 Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 270–279. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.31.

5 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 157–168. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2315.

6 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

8 Jan Dreier and Peter Rossmanith. Approximate evaluation of first-order counting queries. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1720–1739. SIAM, 2021. doi:10.1137/1.9781611976465.104.

9 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:7)2022.

10 Zdenek Dvorák, Daniel Král', and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36:1–36:24, 2013. doi:10.1145/2499483.

11 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

12 Robert Ganian, Petr Hliněný, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of MSO1 model-checking. *Journal of Computer and System Sciences*, 80(1):180–194, 2014.

13 Petr A. Golovach and Yngve Villanger. Parameterized complexity for domination problems on degenerate graphs. In Hajo Broersma, Thomas Erlebach, Tom Friedetzky, and Daniël Paulusma, editors, *Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008, Durham, UK, June 30 - July 2, 2008. Revised Papers*, volume 5344 of *Lecture Notes in Computer Science*, pages 195–205, 2008. doi:10.1007/978-3-540-92248-3_18.

14 Martin Grohe. Generalized model-checking problems for first-order logic. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2001. doi:10.1007/3-540-44693-1_2.

15 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

16 Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 253–266. ACM, 2018. doi:10.1145/3196959.3196970.

17 Henry A. Kierstead and Daqing Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264, 2003. doi:10.1023/B:ORDE.0000026489.93166.cb.

18 Joachim Kneis, Daniel Mölle, and Peter Rossmanith. Partial vs. complete domination: $t$-dominating set. In Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and Frantisek Plasil, editors, *SOFSEM 2007: Theory and Practice of Computer Science, 33rd Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 20-26, 2007, Proceedings*, volume 4362 of *Lecture Notes in Computer Science*, pages 367–376. Springer, 2007. doi:10.1007/978-3-540-69507-3_31.

19 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 189–198. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.39.

20 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005133.

21 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

22 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011. doi:10.1016/j.ejc.2011.01.006.

23 Detlef Seese. Linear time computable problems and first-order descriptions. *Math. Struct. Comput. Sci.*, 6(6):505–526, 1996. doi:10.1017/s0960129500070079.

24 Alexandre Vigny. Dynamic query evaluation over structures with low degree. *CoRR*, abs/2010.02982, 2020. arXiv:2010.02982.

25 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discret. Math.*, 309(18):5562–5568, 2009. doi:10.1016/j.disc.2008.03.024.

# Online Algorithms with Randomly Infused Advice

**Yuval Emek**
Technion, Haifa, Israel

**Yuval Gil**
Technion, Haifa, Israel

**Maciej Pacut**
TU Berlin, Germany

**Stefan Schmid**
TU Berlin, Germany

## Abstract

We introduce a novel method for the rigorous quantitative evaluation of online algorithms that relaxes the "radical worst-case" perspective of classic competitive analysis. In contrast to prior work, our method, referred to as randomly infused advice (RIA), does not make any assumptions about the input sequence and does not rely on the development of designated online algorithms. Rather, it can be applied to existing online randomized algorithms, introducing a means to evaluate their performance in scenarios that lie outside the radical worst-case regime.

More concretely, an online algorithm ALG with RIA benefits from pieces of advice generated by an omniscient but not entirely reliable oracle. The crux of the new method is that the advice is provided to ALG by writing it into the buffer $\mathcal{B}$ from which ALG normally reads its random bits, hence allowing us to augment it through a very simple and non-intrusive interface. The (un)reliability of the oracle is captured via a parameter $0 \leq \alpha \leq 1$ that determines the probability (per round) that the advice is successfully infused by the oracle; if the advice is not infused, which occurs with probability $1 - \alpha$, then the buffer $\mathcal{B}$ contains fresh random bits (as in the classic online setting).

The applicability of the new RIA method is demonstrated by applying it to three extensively studied online problems: paging, uniform metrical task systems, and online set cover. For these problems, we establish new upper bounds on the competitive ratio of classic online algorithms that improve as the infusion parameter $\alpha$ increases. These are complemented with (often tight) lower bounds on the competitive ratio of online algorithms with RIA for the three problems.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online algorithms, competitive analysis, advice

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.44

**Related Version** *Full Version*: https://arxiv.org/abs/2302.05366

**Funding** *Yuval Emek*: VATAT Fund to the Technion Artificial Intelligence Hub (Tech.AI).
*Maciej Pacut*: Austrian Science Fund (FWF) and the German Research Foundation (DFG), grant I 4800-N (ADVISE), 2020-2023.
*Stefan Schmid*: Austrian Science Fund (FWF) and the German Research Foundation (DFG), grant I 4800-N (ADVISE), 2020-2023.

## 1 Introduction

*Competitive ratio* is a widely used metric for evaluating the performance of *online algorithms*. It measures the ratio between the performance of an online algorithm and that of an optimal offline (clairvoyant) algorithm, assuming a worst-case (i.e., adversarial) input sequence. Early on, it has been observed (see, e.g., [53]) that in practice, many online algorithms outperform their theoretical worst-case guarantees. Indeed, in realistic scenarios, the online algorithms tend to "enjoy a good fortune" and rarely encounter the theoretical pitfalls that realize the competitiveness lower bounds (cf. [41]).

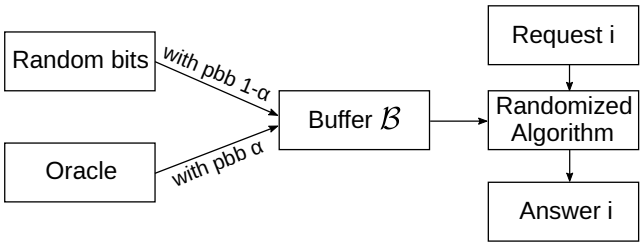




**Figure 1** In each round, the algorithm reads its random bits from buffer $\mathcal{B}$. Under the RIA model, the content of this buffer is replaced by the oracle's advice for that round with probability $\alpha$, independently of other rounds.

This phenomenon has led to extensive research on the analysis of online algorithms beyond the extreme worst-case nature of traditional competitive analysis (see [38] for a recent survey). A prominent approach in this regard is to restrict the power of the adversary that decides on the input sequence, giving rise to the methods of locality of reference [3, 7, 2], access graph [20], smoothed analysis [47, 15], random arrival order [4, 6, 5], independent sampling [27], diffused adversaries [41], and distributional analysis [48, 34]. Another approach is to relax the competitive analysis definition, as done in resource augmentation [50], loose competitiveness [53], and competitiveness with high probability [39]. See also the surveys [29, 23] for additional measures.

In this paper, we wish to advance the study of (randomized) online algorithms beyond worst-case competitive analysis by offering a radically new point of view on the concept of "enjoying a good fortune" (in terms of avoiding the competitiveness pitfalls). Our approach does not restrict the power of the adversary, hence we do not need to justify any assumptions on the request sequence. Moreover, we use the standard definition of competitive analysis (with no relaxations). Last but not least, in contrast to some existing "beyond worst-case" methods, which are limited to certain types of online problems (e.g., locality of reference and access graph), our new method is very general and can be applied to seemingly any online problem.

So, how do we interpret "good fortune" on behalf of a randomized online algorithm ALG without making any assumptions on ALG's input sequence? The answer is simple: we look at the outcome of ALG's random coin tosses. That is, to make ALG more fortunate, all we have to do is to increase the chances of getting good such outcomes.

This raises another question: what makes one outcome of ALG's random coin tosses better than another? To answer this question, we recruit an omniscient *oracle* that generates *advice* for ALG in each round of the execution. The crux of our method, called *randomly infused advice (RIA)*, is that the oracle attempts to write this advice into the buffer $\mathcal{B}$ from which ALG normally reads its random bits. To quantitatively control ALG's good fortune, we introduce an *infusion parameter* $0 \leq \alpha \leq 1$, which determines the probability that the advice is (successfully) infused by the oracle in each round (independently); if the advice is not infused – an event occurring with probability $1 - \alpha$ – then the buffer $\mathcal{B}$ contains fresh random bits (as in the classic online setting). Refer to Figure 1 for an illustration.

We emphasize that the interface between the randomized online algorithm ALG and the oracle is "non-intrusive", i.e., it is defined on top of the standard computational model of (randomized) online algorithms (a.k.a. request-answer games). Therefore, the RIA method is suitable for the analysis of **existing** online algorithms (including classic ones), facilitating the evaluation of their performance beyond the extreme worst-case nature of traditional competitive analysis. This is in contrast to other advice models for online algorithms

(discussed in Section 1.2) in which the oracle-algorithm interface is based on a designated buffer (or tape) from which the algorithm reads the advice. As such, these models require the development of **new**, model-specific, algorithms and cannot be applied to existing ones.

Notice that the RIA model does not impose any limitations on the size of the buffer $\mathcal{B}$, and through it, on the advice size (or the number of random bits) provided to ALG in each round. This raises the concern of making the online algorithm "too powerful" as the (successfully) infused advice may hold excessive information regarding the future requests. To overcome this concern, we restrict our attention to randomized online algorithms which are *randomness-oblivious*, namely, in each round, ALG has access to past requests, past answers, the current request, and the current content of the buffer $\mathcal{B}$ (which contains the current advice or random bits), however ALG cannot access the content of $\mathcal{B}$ in previous rounds. Indeed, all algorithms analyzed in this paper are randomness-oblivious.

The main motivation for studying the RIA method comes from analyzing the performance of randomized online algorithms in scenarios that lie outside the "radical worst-case" regime, assumed in the classic online computation literature. In particular, this new method allows us to compare between different online algorithms that exhibit the same performance guarantees in worst-case scenarios, possibly separating between them in terms of their performance once the scenarios get "a little bit better", and to so without making any explicit assumptions about the request sequence (or the probability distribution thereof).

Another motivation is that the RIA model provides an abstraction for an unreliable predictor (whose role is assumed by the oracle) whose "mistakes" take a random (rather than worst-case) flavor, where the infusion parameter $\alpha$ indicates the (expected) fraction of rounds in which the predictor is correct. In this regard, the non-intrusive interface between the online algorithm and the oracle gives the RIA model a distinctive advantage over existing advice models for online algorithms as it enables the analysis of standard online algorithms in scenarios that include an unreliable predictor, while retaining their worst-case guarantees.

## 1.1 Our Contribution

On top of the conceptual contribution that lies in introducing the RIA model, we make the following technical contribution.

### Upper bounds

The applicability of the new RIA model is demonstrated on three extensively studied online problems: the *paging* problem [50], for which we analyze the classic RandomMark algorithm [32]; the uniform *metrical task system (MTS)* problem [21], for which we analyze the classic UnifMTS algorithm;[1] and the unweighted *online set cover* problem [9], for which we analyze the influential primal-dual algorithm [24, Ch. 4] with randomized rounding (referred to as RandSC). In all cases, our findings are similar to what is called "robustness" and "consistency" in the literature dedicated to online algorithms with predictions [42, 46]: when augmented with RIA, the competitive ratio of these algorithms is never worse than the original, and improves asymptotically as $\alpha \to 1$. Our results are cast in the following three theorems, where we denote the $k$-th harmonic number by $H_k \approx \log k$; we emphasize that in all cases, neither the online algorithm nor the oracle are aware of the infusion parameter $\alpha$.

[1] Due to spatial considerations, the upper bound for uniform MTS is omitted from this version of the paper. Refer to the full version [31] for the upper bound description and analysis.

▶ **Theorem 1.1.** *The competitive ratio of RandomMark augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *on instances of cache size* $k$ *is at most* $\min\{2H_k, \frac{2}{\alpha}\}$.

▶ **Theorem 1.2.** *The competitive ratio of UnifMTS augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *on* $n$*-state instances is at most* $\min\{2H_n, \frac{2}{\alpha} + 2\}$.

▶ **Theorem 1.3.** *The competitive ratio of RandSC augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *on instances with* $n$ *elements and maximum element degree* $d$ *is at most* $O(\min\{\log d \log n, \frac{\log n}{\alpha}\})$.

### Lower bounds

On the negative side, we prove that the upper bound promised in Theorem 1.1 is asymptotically tight for the class of *lazy* algorithms, which are not allowed to change their cache configuration unless there is a page miss.

▶ **Theorem 1.4.** *There does not exist a lazy (randomness-oblivious) online paging algorithm augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *whose competitive ratio on instances of cache size* $k$ *is better than* $\min\{H_k, \frac{1}{\alpha}\}$.

Omitting the restriction to lazy algorithms, we can establish a weaker lower bound.

▶ **Theorem 1.5.** *There does not exist a (randomness-oblivious) online paging algorithm augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *whose competitive ratio on instances of cache size* $k$ *is better than* $\min\{H_k, \frac{1}{k \cdot \alpha}\}$.

The uniform MTS problem generalizes the paging problem on instances that include $n = k + 1$ pages. As Theorems 1.4 and 1.5 hold (already) for such instances, their promised lower bounds are transferred to the uniform MTS problem, where laziness translates to online MTS algorithms that may switch state only when the processing cost is positive [33] (an algorithm class that includes UnifMTS).

▶ **Theorem 1.6.** *There does not exist a lazy (randomness-oblivious) online uniform MTS algorithm augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *whose competitive ratio on* $n$*-state instances is better than* $\min\{H_{n-1}, \frac{1}{\alpha}\}$.

▶ **Theorem 1.7.** *There does not exist a (randomness-oblivious) online uniform MTS algorithm augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *whose competitive ratio on* $n$*-state instances is better than* $\min\{H_{n-1}, \frac{1}{(n-1) \cdot \alpha}\}$.

For online set cover, we establish a lower bound for lazy algorithms, namely, online algorithms which are allowed to buy a set only if it contains the current (uncovered) element (an algorithm class that includes RandSC).

▶ **Theorem 1.8.** *There does not exist a lazy (randomness-oblivious) unweighted online set cover algorithm augmented with RIA with infusion parameter* $0 \leq \alpha \leq 1$ *whose competitive ratio on instances with maximum element degree* $d$ *is better than* $\min\{\frac{1}{2} \log d, \frac{1}{2 \cdot \alpha}\}$.

## 1.2 Novelty and Additional Related Work

### Models of Advice

A well-known and suitable advice model for machine-learned predictions is the model of online algorithms with untrusted advice introduced by Lykouris and Vassilvitskii [42], where the existing literature includes papers on paging [42, 49, 37, 13], metrical task system [11],

and online set cover via the primal-dual approach [12]. In this model, the predictor may be faulty, and the competitive ratio depends on its error so that for low error, the algorithm should perform close to the offline optimum (a.k.a. *consistency*), while even for large error, the algorithm should still fallback to guarantees similar to those of non-augmented online algorithms (a.k.a. *robustness*).

Another well-known advice model is the *perfect* advice model [30, 18] under which many online problems have been studied, including paging, metrical task system [22], and online set cover [28]. In this model, the oracle is fully trustworthy, and its power is therefore quantified via the size (i.e., number of bits) of the advice provided to the online algorithm. This model is related to lookahead [35], where an algorithm is given some number of future requests in advance. The model of perfect advice was later extended to untrusted advice, retaining its focus on measuring the required advice size [10].

Unlike these two advice models, the RIA model does not require any new algorithmic features (e.g., a designated advice tape) and is therefore applicable to existing (standard) online algorithms. Furthermore, our model does not limit the advice size, unlike the perfect advice model, and still allows to arrive at asymptotically tight lower bounds under natural assumptions, in contrast to the machine-learned prediction model where no general lower bounds are known.

#### Online algorithms for paging, MTS, and set cover

Two optimally competitive algorithms for paging are known: PARTITION [43] and EQUITABLE [1]. For the uniform MTS problem, a $(2H_n)$-competitive algorithm was presented in [21], later improved to $H_n + O(\sqrt{\log n})$ in [36]; the latter result nearly matches the $H_n$ lower bound of [21].

For online set cover, the state-of-the-art competitive ratio upper bounds are $O(\log m \log n)$ for the weighted case [9] and $O(\log m \log(n/\mathsf{OPT}))$ for the unweighted case [25], where $m$ and $n$ denote the number of sets (an upper bound on the maximum element degree $d$) and the number of elements, respectively; interestingly, both bounds can be realized by deterministic online algorithms. On the negative side, no (randomized) online algorithm has a competitive ratio better than $\Omega(\log m)$ [40] and no deterministic online algorithm has a competitive ratio better than $\Omega(\log m \log n/(\log\log m + \log\log n))$ [9]. If the (randomized) online algorithm is required to admit a polynomial time implementation, then the competitiveness lower bound improves to $\Omega(\log m \log n)$ assuming that $NP \not\subseteq BPP$ [40].

## 2 Online Algorithms with Randomly Infused Advice

We begin by recalling standard definitions of online algorithms as request-answer games [17]. Our model of online algorithms with randomly infused advice is then defined as a generalization of this model.

### 2.1 Online Algorithms as Request-Answer Games

Consider a finite sequence $\sigma = \langle r_1, \ldots, r_{|\sigma|}\rangle$ of *requests*, where each request $r_i$ is taken from a set $\mathcal{R}$. A *solution* for $\sigma$ is a sequence $\lambda = \langle a_1, \ldots, a_{|\sigma|}\rangle$ of *answers*, where each answer $a_i$ is taken from a set $\mathcal{A}$. For a given minimization problem, the quality of a solution $\lambda$ for a request sequence $\sigma$ is determined by means of a *cost function* $f : \mathcal{R}^{|\sigma|} \times \mathcal{A}^{|\sigma|} \to \mathbb{R} \cup \{\infty\}$.[2] Let $\mathsf{OPT}(\sigma) = \inf_{\lambda \in \mathcal{A}^{|\sigma|}} f(\sigma, \lambda)$ denote the cost of an *optimal solution* for $\sigma$.

[2] We restrict our attention to minimization problems as these are the problems addressed in the current paper. Extending our setting to maximization problems is straightforward.

In the realm of online algorithms, the requests are revealed one-by-one, in discrete *rounds*, so that upon receiving request $r_i$ in round $i$, a (randomized) online algorithm ALG outputs the (random) answer $a_i$ irrevocably. That is, the solution $\lambda_{\mathsf{ALG}} = \langle a_1, \ldots, a_{|\sigma|} \rangle$ produced by ALG is defined so that each answer $a_i$ is computed as a function of (1) the request subsequence $r_1, \ldots, r_i$; (2) the answer subsequence $a_1, \ldots, a_{i-1}$; and (3) round $i$'s random bit string $\mathcal{B}_i \in_R \{0,1\}^L$, where the parameter $L \in \mathbb{Z}_{\geq 0}$ is specified by the algorithm's designer (possibly as a function of the parameters of the problem).[3]

The performance of an online algorithm ALG is measured via competitive analysis: we say that ALG is *c-competitive* if there exists a constant $b$ (that may depend on the parameters of the problem) such that $\mathbb{E}[\mathsf{ALG}(\sigma)] \leq c \cdot \mathsf{OPT}(\sigma) + b$ for any request sequence $\sigma$, where $\mathsf{ALG}(\sigma)$ is the random variable that takes on the cost of the solution produced by ALG in response to a request sequence $\sigma$. The request sequence $\sigma$ is assumed to be determined by a malicious *adversary*; we stick to the convention of an *oblivious adversary* [19, Ch. 4] which means that the adversary knows ALG's description, but is unaware of the outcome of ALG's random coin tosses.

## 2.2 Randomly Infused Advice

In this paper, we introduce an extension of online algorithms, referred to as online algorithms with *randomly infused advice (RIA)*. In the RIA model, an algorithm ALG is assisted by a powerful, yet not entirely reliable, *oracle* that has access to the entire request sequence $\sigma$. Formally, for any request sequence $\sigma = \langle r_1, \ldots, r_{|\sigma|} \rangle$ and round $1 \leq i \leq |\sigma|$, the oracle $\mathcal{O}$ is defined by an *advice function* $\mathcal{O}_{\sigma,i} : \mathcal{A}^{i-1} \to \{0,1\}^L$ that maps each answer subsequence $\langle a_1, \ldots, a_{i-1} \rangle$ to a bit string $\mathcal{O}_{\sigma,i}(a_1, \ldots, a_{i-1}) \in \{0,1\}^L$, referred to as the round $i$'s *advice*. Notice that the length of the advice bit string is equal to the length $L$ of ALG's random bit string.

The RIA model is associated with an *infusion parameter* $0 \leq \alpha \leq 1$ that quantifies the (un)reliability of the oracle $\mathcal{O}$. Specifically, in each round $i$, the bit string $\mathcal{B}_i$ (provided to the online algorithm in that round) is now determined based on the following random experiment (independently of the other rounds): with probability $\alpha$, the round $i$'s advice is *infused* into $\mathcal{B}_i$, that is, $\mathcal{B}_i \leftarrow \mathcal{O}_{\sigma,i}(a_1, \ldots, a_{i-1})$; with probability $1 - \alpha$, the bit string $\mathcal{B}_i$ is picked uniformly at random, that is, $\mathcal{B}_i \in_R \{0,1\}^L$.

In other words, in each round $i$ where the infusion is successful (an event occurring with probability $\alpha$), the oracle's advice "smoothly" substitutes the random bit string $\mathcal{B}_i$ before it is provided to ALG; if the infusion is not successful, then $\mathcal{B}_i$ remains a random bit string. We emphasize that ALG and $\mathcal{O}$ are not aware (at least not directly) of whether the advice is successfully infused in the round $i$, nor are they aware of the infusion parameter $\alpha$ itself.

The competitive ratio of online algorithms ALG with RIA is typically expressed as a function of the infusion parameter $\alpha$, where the extreme case of $\alpha = 0$ corresponds to standard online computation (with no advice). The ultimate goal is to provide guarantees on the competitiveness of ALG for any $0 \leq \alpha \leq 1$.

[3] We use a single parameter $L$ (that is often kept implicit in the online algorithm's description) for simplicity of the exposition; it can be easily generalized to a (not necessarily bounded) sequence $L_1, L_2, \ldots$ of round-dependent parameters.

## 2.3 Randomness-Oblivious Online Algorithms

Recall that the aforementioned definition of online algorithms dictates that when the online algorithm ALG determines the answer $a_i$ associated with round $i$, it is aware of the requests $r_{i'}$ and answers $a_{i'}$ associated with past rounds $i' < i$, as well as the request $r_i$ and random bit string $\mathcal{B}_i$ associated with the current round $i$, however it is not aware (at least not directly) of the random bit strings $\mathcal{B}_{i'}$ associated with past rounds $i' < i$. This model choice is made to prevent an online algorithm ALG with RIA from passing information received through the (successfully infused) advice to future rounds, thus over-exploiting the lack of an explicit (model specific) bound on the length of the random / advice bit strings. To distinguish the online algorithms that adhere to this formulation from general online algorithms (that may maintain a persistent memory that encodes past random bits), we refer to the former as *randomness-oblivious* online algorithms.

# 3 Paging

In the *online paging* problem [50], we manage a two-level memory hierarchy, consisting of a slow memory that stores the set of all $n$ pages, and a fast memory, called the *cache*, that stores any size $k$ subset of pages. We are given a sequence $\sigma$ of requests to the pages. If a requested page is not in the cache, a *page fault* occurs, and the page must be moved to the cache. Since the cache is limited in size, we must specify which page to evict to make space for the requested page. The goal is to minimize the number of page faults.

In this section, we analyze an elegant randomized online algorithm RandomMark, introduced by Fiat, Karp, Luby, McGeoch, Sleator and Young [32], in the randomly infused advice framework. The algorithm RandomMark maintains a bit associated with each page in the cache. Initially the bits of all pages are set to 0 (the pages are *unmarked*), and after requesting a page, we bring it to the cache if it is not in the cache yet, and we set its bit to 1 (we *mark* the page). To bring a page to the cache, we may need to evict another page to make space for it. In such a case, RandomMark evicts a page uniformly at random chosen from the unmarked pages. If no unmarked page exists, we unmark all pages. This strategy has been shown to be $2H_k$-competitive [32], where $H_k$ is the harmonic number, and no randomized algorithm can be better than $H_k$-competitive.

## 3.1 RandomMark With Infused Advice

With help of randomness, the classic RandomMark decides on the final candidate to evict: a random node among unmarked pages. With infused advice, in some rounds the randomness source used by RandomMark contains advice instead of random bits. The presence of clairvoyent advice brings obvious advantages, but also brings challenges: not all pages can be evicted, only the unmarked ones.

### Unmarked Longest-Forward-Distance Oracle

An optimal offline algorithm for paging is to evict the item with the access time furthest in the future [16], also known as *longest forward distance* (LFD) algorithm. However, we cannot directly design an oracle for RandomMark around LFD, as it may advise to evict a marked page, but RandomMark never evicts marked pages. Hence, we propose a variant of this algorithm that can act as an oracle for RandomMark. Such an oracle, denoted $O_{ULFD}$, advises RandomMark to evict the page with the longest forward distance *among the unmarked items* of RandomMark.

### Analysis of RandomMark

How well can RandomMark perform with infused advice? To find out, we consider the RandomMark algorithm assisted with the oracle $O_{ULFD}$, and we express the algorithm's competitive ratio of in terms of the infusion parameter $\alpha$ (the probability of receiving advice in each round). Later in this paper, we will show that RandomMark with $O_{ULFD}$ is asymptotically optimal (Theorem 5.4).

▶ **Theorem 3.1.** *The competitive ratio of RandomMark with the oracle $O_{ULFD}$ with RIA on instances of cache size $k$ (against the oblivious adversary) is at most $\min\{2H_k, \frac{2}{\alpha}\}$, where $H_k$ is the $k$-th harmonic number, and $0 \leq \alpha \leq 1$ is the infusion parameter.*

Before proving this theorem, we recall the definition of a $k$-phase partitioning of an input sequence, and we derive sufficient conditions to stop incurring further page faults in a phase.

We begin by recalling basic definitions from the analysis of RandomMark by [32]. We consider the *$k$-phase partition* of the input sequence $\sigma$, following the notation from [19]: phase 0 is the empty sequence, and each phase $i > 0$ is the maximal sequence following the phase $i-1$ that contains at most $k$ distinct page requests since the start of the $i$th phase. In a phase of any marking algorithm, a page requested in the phase is *stale* if it is unmarked but was marked in the previous phase, and a page is *clean* if it is neither stale nor marked.

In addition to these standard definitions, we define the set of *vanishing pages* as the set of the pages requested in the previous phase, but not in the current phase. We claim that after evicting all vanishing pages, marking algorithms incur no further cost in the phase, since a configuration is reached where all the remaining requests in the current phase are free (page hits).

▶ **Lemma 3.2.** *Fix an input sequence $\sigma$, consider its $k$-phase partition, and fix any phase $P$ that is not the first or the last phase. Then, (1) we have exactly $c$ vanishing pages, where $c$ is the number of clean pages in the phase; and (2) after evicting all vanishing pages, no marking algorithm for paging incurs further cost in the phase.*

**Proof.** In the phase $P$, we have exactly $k$ requests to distinct pages: to $k-c$ stale pages and to $c$ clean pages. Only the clean pages can replace the vanishing pages, hence we have exactly $c$ vanishing pages. Hence, the first claim holds.

If at any point all $c$ vanishing pages are evicted, this means that all $c$ clean pages were requested in the phase already. The remaining requests in the phase can concern only stale pages. As no vanishing pages remain in the cache, the cache consists of $c$ clean pages and $k-c$ stale pages. Hence, after evicting all vanishing pages, any marking algorithm incurs no further cost in the phase, and the second claim holds. ◀

Finally, we prove our main claim for paging: RandomMark is $\min\{2H_k, \frac{2}{\alpha}\}$-competitive. We repeat the classic arguments of [32] to arrive at the bound $2H_k$, and we analyze the offline algorithm *unmarked longest forward distance*, employed by the oracle that probabilistically interacts with the oracle, to arrive at the bound $\frac{2}{\alpha}$.

**Proof of Theorem 3.1.** Fix any input sequence $\sigma$ and consider its $k$-phase partition. Consider any phase that is not the first or the last one. Let $c$ be the number of clean pages in the phase.

We claim that the expected number of page faults is upper bounded by $c/\alpha$. If the algorithm incurs a page fault, and it receives the oracle's advice, and there are still some vanishing pages in the cache, then the algorithm evicts a vanishing page; this follows since

the vanishing pages are not requested in the current phase, hence they have larger forward distance than other stale pages, and the vanishing pages are unmarked. By Lemma 3.2, evicting all vanishing pages means that no further cost is incurred throughout the phase, hence the number of page faults in the phase is upper bounded by the number of page faults until the algorithm receives $c$ rounds of advice from the oracle (not necessarily consecutive). The expected number of page faults until receiving $c$ rounds of advice is $c/\alpha$, since this is the expected number of independent tosses of $\alpha$-biased coin until getting $c$ heads outcomes.

Next, we repeat the classic arguments of [32]: the expected number of page faults of the algorithm is also upper bounded by $c \cdot H_k$. Consider an $i$-th request to a stale page in the phase for $i = 1, 2, 3, \ldots, s$. Let $c(i)$ denote the number of clean pages requested in the phase immediately before the $i$-th request to a stale page, and let $S(i)$ denote the set stale pages that remain in the cache before the $i$-th request to a stale page, and let $s(i) = |S(i)|$. For $i = 1, 2, 3, \ldots, s$, we compute the expected cost of the $i$-th request to a stale page. When the algorithm serves the $i$-th request to a stale page, exactly $s(i) - c(i)$ of the $s(i)$ stale pages are in the cache. The stale pages are in the cache with equal probability, say $p$, since these are never evicted with the help of advice, but are evicted uniformly from unmarked pages when a page fault occurs in rounds without advice. The vanishing pages are in the cache with probability at most $p$, since they can be evicted both in the rounds with and without the advice. For the all $s(i)$ stale pages the probability of being in the cache sums to 1, hence $p \leq 1/s(i)$. Fix a request to a stale page. The page is in the cache with probability $(s(i) - c(i)) \cdot p$, hence the expected cost of the request is

$$1 - (s(i) - c(i)) \cdot p \leq 1 - \frac{s(i) - c(i)}{s(i)} = \frac{c(i)}{s(i)} \leq \frac{c}{k - i + 1}.$$

Hence, the total cost of the request to the stale pages is $\sum_{i=1}^{s} c/(k - i + 1) \leq \sum_{i=2}^{k} c/i = c \cdot (H_k - 1)$. The total cost in the phase includes the cost of serving the clean page and stale pages, in total $c \cdot H_k$.

We conclude that the number of page faults of the algorithm in a phase is upper-bounded by both $c \cdot H_k$ and $c/\alpha$. By arguments of [32, Theorem 1], the amortized number of faults made by OPT during the phase is at least $c/2$. Summing over all phases but the first and the last one, the competitive ratio is at most $\min\{2H_k, \frac{2}{\alpha}\}$. The first and the last phase incurs cost bounded by $2k$, which we account in the additive in the competitive ratio. ◀

The above analysis is asymptotically tight with the lower bound given in Theorem 5.3. However, for the special case $n = k + 1$, the result is tight: the competitive ratio of RandomMark with the oracle $O_{ULFD}$ is $\min\{H_k, \frac{1}{\alpha}\}$, since in each phase but the last phase, any offline algorithm pays at least 1, and the number of clean pages is also 1.

The algorithm RandomMark with perfect advice ($\alpha = 1$) is equivalent to an offline algorithm that evicts the unmarked item with the longest forward distance. The Theorem 3.1 implies that this algorithm is optimal for $n = k + 1$, and a 2-approximation for any $n$.

## 4 Set Cover

In the *set cover* problem, we are given a universe $\mathcal{U}$ of $n$ elements and a set $\mathcal{F} = \{S_1, \ldots, S_m\}$ of $m$ subsets $S_1, \ldots, S_m \subseteq \mathcal{U}$ such that $S_1 \cup \cdots \cup S_m = \mathcal{U}$. For each element $e \in \mathcal{U}$, let $\mathcal{F}(e) = \{S \in \mathcal{F} \mid e \in S\}$ be the collection of sets that cover it. In the online setting, a subset

$\mathcal{U}' \subseteq \mathcal{U}$ of elements arrive one by one in an arbitrary order.[4] Upon the arrival of an element $e$, the algorithm is required to cover it (i.e., if $e$ was not previously covered by the algorithm, then the algorithm must select a set from $\mathcal{F}(e)$). We emphasize that the algorithm does not know $\mathcal{U}'$ (or its size) in advance and that any previously selected set cannot be removed from the solution obtained by the online algorithm. The cost of a solution to the set cover problem is the number of sets selected.

In the standard linear program (LP) relaxation for set cover, each set $S \in \mathcal{F}$ is associated with a variable $x_S$. The objective is to minimize the sum $\sum_{S \in \mathcal{F}} x_S$ subject to the constraints $\sum_{S \in \mathcal{F}(e)} x_S \geq 1$ for each element $e \in \mathcal{U}'$, and $x_S \geq 0$ for all $S \in \mathcal{F}$.

Recall that in the context of set cover in the RIA model, we focus on *lazy* algorithms, i.e., algorithms that adhere to the following restrictions upon the arrival of an elemnt $e$: (1) if $e$ is already covered by the algorithm, then in the current round the algorithm does not select any additional sets to its solution; and (2) if $e$ is not covered yet, then in the current round the algorithm may only select sets from $\mathcal{F}(e)$. Notice that this restriction prevents the trivial oracle strategy of simply advising to select all the sets of an optimal set cover at each round.

We describe an online algorithm with RIA for set cover in three stages. First, we present an algorithm that obtains a fractional solution $\mathbf{x}$ to the relaxed LP. Then, we present an online randomized rounding scheme that can be incorporated into the fractional set cover algorithm to obtain an integral solution which is feasible with high probability. Finally, we present the oracle's advice.

### Fractional set cover algorithm

We use the basic discrete algorithm presented by Buchbinder and Naor in [24, Chapter 4.2, Algorithm 1].[5] The algorithm operates as follows. Initially, set $x_S = 0$ for all $S \in \mathcal{F}$. Upon arrival of an element $e$, if $\sum_{S \in \mathcal{F}(e)} x_S < 1$, then update $x_S \leftarrow 2 \cdot x_S + 1/|\mathcal{F}(e)|$ for all $S \in \mathcal{F}(e)$. Observe that at the end of the round, it is guaranteed that the fractional primal solution maintained by the algorithm satisfies the constraint since the algorithm adds at least $1/|\mathcal{F}(e)|$ to the variable $x_S$ for each set $S \in \mathcal{F}(e)$.

Let $d = \max_{e \in \mathcal{U}'} |\mathcal{F}(e)|$ be the maximum degree of an element. The following assertion on the competitive ratio is established by Buchbinder and Naor in [24].

▶ **Lemma 4.1** ([24]). *The fractional set cover algorithm is $O(\log d)$-competitive.*

### Randomized rounding

An online rounding scheme that randomly obtains an integral solution from the fractional set cover algorithm was constructed by Alon et al. in [8]. The solution produced by the rounding scheme of [8] is feasible with high probability while incurring a multiplicative factor of $O(\log n)$ to the expected cost. However, this rounding method does not fit our advice framework. This is because all random coins are tossed in the beginning to compute a threshold for each set. Thus, we present a slightly different rounding method that fits our framework while maintaining similar guarantees.

The rounding procedure operates as follows. Consider an element $e$ and let $\mathbf{x}$ and $\mathbf{x}_{int}$ be the solution maintained by the fractional algorithm and the (integral) solution maintained by the rounding scheme, respectively, at the time of $e$'s arrival. If $e$ is already covered

[4] While our results in the current section are expressed in terms of the size of the universe $n$, it can be modified to obtain the same asymptotic bounds in terms of the length of the element sequence $|\mathcal{U}'|$.

[5] We note that the algorithm presented in [24] is designed for weighted set cover. The algorithm presented in this paper is its application for the case of unit weights.

by either the current fractional solution or the current integral solution produced by the rounding, then we do nothing (we will later show that the feasibility of $\mathbf{x}_{int}$ is maintained with high probability in this case). Otherwise ($e$ is not covered by both solutions), we update $\mathbf{x}$ according to the fractional algorithm. For each $S \in \mathcal{F}(e)$, let $x_S^{beg}$ be the value of the variable $x_S$ at the beginning of the round and let $\delta(S) = x_S^{beg} + 1/|\mathcal{F}(e)|$ be the additive increase to $x_S$ that occurs during the round. The rounding is obtained by independently selecting each set $S \in \mathcal{F}(e)$ to the cover with probability $\min\{1, \delta(S) \cdot \Theta(\log n)\}$.

We refer to the randomized algorithm described above (i.e., the fractional set cover algorithm combined with the rounding scheme) as RandSC. The properties of RandSC are described in the following lemma.

▶ **Lemma 4.2.** *RandSC is $O(\log n \log d)$-competitive and computes a feasible solution with high probability.*[6]

**Proof.** Let $\mathbf{x}$ be the solution obtained by the fractional algorithm at termination. Recall that in each round, set $S$ is selected with probability at most $\delta(S) \cdot c \log n$ (for a constant $c > 0$). By linearity of expectation, the total expected cost associated with $S$ is $O(\log n) \cdot x_S$. Thus, the expected cost of RandSC is $O(\log n) \cdot \sum_{S \in \mathcal{F}} x_S = O(\log n \log d) \cdot \mathsf{OPT}$.

We now bound the probability that there exists an element that was not covered by the integral solution produced by RandSC when it arrived. Consider an element $e'$ arriving at round $r$. Notice that by construction, $e'$ must be covered by the fractional solution at the end of round $r$. We argue that this implies that $e'$ is covered by the integral solution with high probability. Let $\ell = |\mathcal{F}(e')|$ and let $S^1, \ldots S^\ell$ denote the sets in $\mathcal{F}(e')$. Let us denote by $\delta_{i,j}$ the increase to the variable $x_{S^i}$ associated with set $S^i$ in round $j$ and let $p_{i,j}$ the probability that $S^i$ was selected to the integral solution at round $j$. If $p_{i,j} = 1$ for some $i \leq \ell$ and $j \leq r$, then $e'$ is covered by the end of round $r$ with probability 1. Otherwise, due to the independence of selection events, the probability that $e'$ is not covered by the integral solution at the end of round $r$ is

$$\prod_{i=1}^{\ell} \prod_{j=1}^{r} (1 - p_{i,j}) \leq e^{-\sum_{i=1}^{\ell} \sum_{j=1}^{r} p_{i,j}} = e^{-c \log n \sum_{i=1}^{\ell} \sum_{j=1}^{r} \delta_{i,j}} \leq n^{-c},$$

where the final inequality holds because the fractional algorithm guarantees that $e'$ is covered at round $r$ and thus $\sum_{i=1}^{\ell} \sum_{j=1}^{r} \delta_{i,j} \geq 1$. By a union bound argument, the probability that there exists a set that is not covered by the integral solution is at most $n^{1-c}$. Thus, RandSC produces a feasible solution with probability at least $1 - 1/n^{c-1}$. ◀

## Oracle's advice

The idea of the oracle's advice is to boost the probability of selecting "good" sets while not losing the probabilistic feasibility guarantee of Lemma 4.2. For the sake of analysis, let us assume that the oracle is randomized (observe that this assumption does not enhance the oracle's power since the oracle can deterministically compute an optimal realization of the randomized selection). Let $\mathcal{A}^* \subseteq \mathcal{F}$ be an optimal solution for the set cover instance. Consider the arrival of an element $e$ that was not covered yet by both the fractional and integral solutions and let $p_S$ be the probability that set $S$ is selected in the current round of

[6] For simplicity, RandSC is described as a Monte Carlo algorithm. It can be easily transformed into a Las Vegas algorithm as follows: whenever an element $e$ is not covered by RandSC upon the end of a round, select an arbitrary set that covers $e$ into the solution. Notice that the added expected cost is negligible.

RandSC for each set $S \in \mathcal{F}(e)$. The oracle's advice is as follows: (1) each set $S \in \mathcal{F}(e) \cap \mathcal{A}^*$ is selected to the advice; and (2) each set $S \in \mathcal{F}(e) - \mathcal{A}^*$ is independently selected to the advice with probability $p_S$. Notice that the argument used in Lemma 4.2 regarding the feasibility of the solution still holds since the oracle does not decrease the selection probability of any set at a given round. Denoting this oracle by $O_{boost}$, we can establish the following theorem.

▶ **Theorem 4.3.** *The competitive ratio of RandSC with the oracle $O_{boost}$ against an oblivious adversary is $O(\log n) \cdot \min\{1/\alpha, \log d\}$, where $0 \leq \alpha \leq 1$ is the infusion parameter.*

**Proof.** We start by showing that RandSC with $O_{boost}$ is $O(\log n \log d)$-competitive. Notice that by Lemma 4.2, the total expected cost associated with sets $S \in \mathcal{F} - \mathcal{A}^*$ is $O(\log n \log d) \cdot \mathsf{OPT}$. In addition, the total cost of sets in $\mathcal{A}^*$ is bounded by $|A^*| = \mathsf{OPT}$. Therefore, the expected cost of the solution produced by RandSC with $O_{boost}$ is $O(\log n \log d) \cdot \mathsf{OPT}$.

We now show that RandSC with $O_{boost}$ is $O(\frac{\log n}{\alpha})$-competitive. Consider the run of RandSC with $O_{boost}$ on some element sequence. We refer to a round as a selection round if there exists a set that is selected with a positive probability in that round. Notice that we can bound the cost of RandSC with $O_{boost}$ only in selection rounds (for non-selection rounds no cost is incurred). Observe that in each selection round, the probability of selecting a set from $\mathcal{A}^*$ is at least $\alpha$ (the probability of receiving advice). Moreover, if at some point in the execution all sets from $\mathcal{A}^*$ were selected, then there are no selection rounds after that point (since $\mathcal{A}^*$ covers all elements). Hence, the expected number of selection rounds during the execution is at most $|\mathcal{A}^*|/\alpha$.

To complete our analysis, we argue that the expected cost associated with sets that are not in $\mathcal{A}^*$ at each selection round is $O(\log n)$. Consider a selection round in which an elements $e$ arrived. Recall that for each set $S \in \mathcal{F}(e) - \mathcal{A}^*$, we define $\delta(S) = x_S^{beg} + 1/|\mathcal{F}(e)|$, where $x_S^{beg}$ is the value of variable $x_S$ at the beginning of the round, and select each set $S \in \mathcal{F}(e) - \mathcal{A}^*$ to the cover with probability $\min\{1, \delta(S) \cdot \Theta(\log n)\}$. Thus, the total expected cost that comes from the sets $S \in \mathcal{F}(e) - \mathcal{A}^*$ in the round is bounded by $O(\log n) \cdot \sum_{S \in \mathcal{F}(e) - \mathcal{A}^*} x_S^{beg} + \frac{1}{|F(e)|} \leq O(\log n) \cdot 2 = O(\log n)$. Since the total cost associated with sets from $\mathcal{A}^*$ is at most $|\mathcal{A}^*|$, we get that the total expected cost of RandSC with $O_{boost}$ is $O(\log n) \cdot |\mathcal{A}^*|/\alpha = O(\frac{\log n}{\alpha}) \cdot \mathsf{OPT}$. ◀

# 5 Lower Bounds

In this section we show fundamental limitations of online algorithms with RIA. First, we give a lower bound for competitiveness with RIA for online set cover, under the assumption that the algorithm is lazy (buys sets only when they are needed to cover the current element). Second, we give a lower bound for competitiveness with RIA for paging, that we improve to an asymptotically tight lower bound for the case of lazy algorithms. The lower bound for paging implies the lower bound for the uniform metrical task system.

## 5.1 Online Set Cover

We give a lower bound for the competitive ratio of any online randomized algorithm with RIA for online set cover. The construction of the input sequence is similar to the lower bounds given in [40, Theorem 2.2.1] and [24, Lemma 4.6]. The bound is given for randomness-oblivious (defined in Section 2.3) and lazy algorithms (lazy algorithms are allowed to buy a set only if it contains the current element).

▶ **Theorem 5.1.** *Assume that an online randomized algorithm with RIA for online set-cover is lazy, randomness-oblivious and strictly $c$-competitive against the oblivious adversary. Then $c \geq \min\{\frac{1}{2}\log n, \frac{1}{2\alpha}\}$, where $n$ is the size of the universe of element, and $\alpha$ is the infusion parameter.*

**Proof.** Fix any lazy, randomness-oblivious online randomized algorithm ALG with RIA, its oracle $O$ and the infusion parameter $\alpha$. The adversary is oblivious to random choices of the algorithm, but it has access to the description of the algorithm, the oracle and the infusion parameter, hence can maintain the probability distribution of ALG's cache configurations.

Consider a complete binary tree with $d$ leaves. The items to be covered are the nodes of the tree, and the sets are the $d$ root-leaf paths. Our sequence $\sigma$ will be the items on one root-leaf path, starting from the root and going downward.

We chose the sequence of items to request corresponding to a path in the complete binary tree as follows. Let $F(e)$ be the family of sets that cover the item $e$, and let $p_S$ be the probability that ALG currently has the set $S$ in the solution. The first request is to the root of the tree. For the $i$-th request, we choose one of the children, $x$ or $y$ of the item requested in the $(i-1)$-th request, depending on the probability distribution of the sets that cover these items. To decide between $x$ and $y$, we choose the item $r \in \{x, y\}$ with no smaller sum of the probability mass $\sum_{F(x)} p_S$.

We consider two cases depending on whether or not the algorithm received advice for $\sigma$.

1. Assume the algorithm did not receive advice for $\sigma$. In such case, the algorithm acts as an online algorithm without advice. Notice that the total probability mass of sets that do not appear in subsequent iterations add up to at least $1/2$. Each path has length $\log n$, and the algorithm pays at least $\frac{1}{2}$ for each such round, hence overall the algorithm pays $\frac{1}{2}\log d$.
2. Assume the algorithm received advice for $\sigma$. In expectation, the number of rounds before getting advice is $\frac{1}{\alpha}$, and the algorithm pays at least $\frac{1}{2}$ for each such round, hence in total the algorithm pays at least $\frac{1}{2} \cdot \frac{1}{\alpha} = \frac{1}{2\alpha}$.

Note that $\sigma$ can be covered by a single set, namely the one that corresponds to the leaf where the path ends, hence $\mathsf{OPT}(\sigma) = 1$. The online algorithm pays at least $\min\{\frac{1}{2}\log d, \frac{1}{\alpha}\}$ for any sequence $\sigma$ of the form described above, hence ALG is at least strictly $\min\{\frac{1}{2}\log d, \frac{1}{2\alpha}\}$-competitive. ◀

For lazy algorithms, we can obtain a lower bound in terms of the number of $d$. We say that an online algorithm for online set cover is *lazy* if it buys a set only if the current element is not yet covered, and then it may buy only sets that cover the current element. The next bound is stronger than the previous one, as it the bound is on the competitive ratio in the classic sense, with the possible additive constant, as opposed to the previous bound on the strict competitiveness.

▶ **Theorem 5.2.** *Assume that an online randomized algorithm with RIA for online set-cover is lazy, randomness-oblivious and $c$-competitive against the oblivious adversary. Then $c \geq \min\{\frac{1}{2}\log d, \frac{1}{2\alpha}\}$, where $d$ is the maximum element degree, and $\alpha$ is the infusion parameter.*

**Proof.** We repeat the construction from the previous proof of Theorem 5.1 in phases, in each phase using a binary tree of $2d$ items.

As the algorithm is lazy, it cannot buy sets from future phases, and the sets used in different phases are disjoint, hence advice received in any phase cannot decrease the cost of the algorithm in any future phase.

Fix any phase. We consider two cases depending on whether or not the algorithm received advice in this phase. If the algorithm received advice, then it pays at least $\frac{1}{2} \cdot \frac{1}{\alpha} = \frac{1}{2\alpha}$, as the expected number of rounds in this phase before receiving advice concerning sets in this phase is $\frac{1}{\alpha}$. Otherwise, if the algorithm did not receive advice, then it pays at least $\frac{1}{2} \cdot \log d$, following the arguments from the previous proof.

In total, the algorithm pays at least $\min\{\frac{1}{2}\log d, \frac{1}{2\alpha}\}$ in each phase, and an optimal algorithm can cover the items in each phase using a single set, hence the algorithm is at least $\min\{\frac{1}{2}\log d, \frac{1}{2\alpha}\}$-competitive.

Note that we can repeat this construction arbitrary number of iterations to obtain a lower bound on the competitive ratio, as opposed to a lower bound on strict competitive ratio. In each iteration, we use a new set of items and sets corresponding to a binary balanced tree, and the maximum number of sets that cover any item $d$ does not increase by repeating the construction. Hence, no randomized algorithm with infused advice can be better than $\min\{\frac{1}{2}\log d, \frac{1}{2\alpha}\}$-competitive. ◀

## 5.2 Paging and Metrical Task Systems

In this section we give a lower bound for competitiveness of randomized online algorithms with RIA for paging. The uniform metrical task system problem generalizes the paging problem on instances that include $n = k + 1$ pages, hence the lower bound for paging is a common lower bound for paging and uniform metrical task system. We restrict our attention to randomness-oblivious algorithms, as defined in Section 2.3. Our lower bound for any randomness-oblivious algorithm is loose by a factor of $1/k$; but with the natural assumption that the algorithm is *lazy*, we get rid of the $1/k$ factor, and for lazy algorithms the upper bounds for paging (Theorem 3.1) and uniform metrical task systems (Theorem 11 in the full version [31]) are asymptotically optimal.

To show the lower bound in this section, we apply Yao's Minimax Principle [52] to competitiveness of randomized online algorithms. In the case of classic online algorithms, the lower bound for the competitiveness of the best *deterministic* online algorithm on a distribution of inputs implies a lower bound on the competitiveness of any randomized online algorithm on any input sequence.

We define a deterministic equivalent of algorithms with RIA. To this end, we add to each request the information whether the request is served by a deterministic online algorithm or by the oracle. We will analyze performance of such an algorithm on a distribution of requests, where each round is served by the algorithm with probability $1 - \alpha$, and by the oracle with probability $\alpha$. To give a lower bound for randomness-oblivious algorithms (as defined in Section 2.3), we need to define a deterministic equivalent of such algorithms that we refer to as *deterministic advice-oblivious* algorithms: the answer for each request not served by the oracle is determined by the current request, previous requests and previous answers.

To apply Yao's priciple to competitiveness of randomized online algorithms with RIA, we construct a matrix representation of the game, where the row player corresponds to a deterministic advice-oblivious algorithm combined with the offline oracle algorithm, and the column player represents the adversary who specifies the input sequence. The value in each row-column pair of the matrix equals the expected cost incurred by the algorithm-oracle pair on the input sequence, divided by the cost of an optimal offline solution for the input sequence. The choice of whether the online algorithm or the oracle serves a request is beyond the control of both the adversary and the online algorithm, and to compute the value for a row-column pair we take the expectation over all possibilities where for each request independently, the deterministic algorithm serves the request with probability $1 - \alpha$, and the oracle serves the request with probability $\alpha$. Notably, a randomness-oblivious algorithm is no more powerful than a distribution over the deterministic advice-oblivious algorithms.

▶ **Theorem 5.3.** *Assume that an online randomized algorithm with RIA for online paging is randomness-oblivious and c-competitive against the oblivious adversary. Then* $c \geq \min\{H_k, \frac{1}{k\cdot\alpha}\}$, *where* $\alpha$ *is the infusion parameter.*

**Proof.** To prove the theorem, we apply Yao's Minimax Principle [52] to competitiveness of randomized algorithms. Consider any deterministic advice-oblivious algorithm $A$ for paging, and construct the following distribution over input sequences. Each round is served by $A$ with probability $1-\alpha$, and by the oracle with probability $\alpha$. The distribution over requests to pages is constructed as follows. Let $S = \{p_1, p_2, p_3, \ldots, p_{k+1}\}$ be a set of $k+1$ pages. We construct a probability distribution for choosing a request sequence. The first request $\sigma(1)$ is chosen uniformly at random from $S$. Every other request $\sigma(t)$, $t > 1$, is made to a page that is chosen uniformly at random from $S \setminus \{\sigma(t-1)\}$. A phase starting with $\sigma(i)$ ends with $\sigma(j)$, where $j, j > i$ is the smallest integer such that $\{\sigma(i), \sigma(i+1), \ldots, \sigma(j)\}$ contains $k+1$ distinct pages.

We claim that for any advice-oblivious algorithm, the advice received in past phases cannot reduce the cost of the algorithm in future phases. We argue as follows. First, the advice-oblivious algorithm is forbidden to store past advice in its internal memory for future use. Second, no algorithm can store meaningful advice for the future in its cache configuration: each phase contains requests to $k+1$ different items, so for any cache configuration at the start of the phase, there is always at least 1 *clean page*: a page that is requested in the phase that the algorithm does not have in the cache at the start of the phase.

In our bounds, we use that the average cost of the algorithm for each request is $1/k$; this follows because the requested page is random and each of its pages is outside the cache with equal probability.

We lower-bound the cost of the algorithm in each phase in two ways, depending on whether or not the algorithm receives advice in any round of the phase.

1. Assume that the algorithm does not receive advice in any round of the phase. In such case, the algorithm acts as an online algorithm without advice throughout the phase, and the expected cost of the algorithm in the phase is at least $H_k$, following the standard arguments [44]: the expected length of the phase is $k \cdot H_k$, the average cost of the algorithm for each request is $1/k$, therefore the cost of the algorithm within a phase is at least $H_k$.
2. Assume that the algorithm receives advice in some round of the phase. To receive advice, we need in expectation $1/\alpha$ rounds prior to the advice round. The average cost of the algorithm for each request is $1/k$, hence the expected cost is at least $\frac{1}{k \cdot \alpha}$.

An optimal offline algorithm OPT incurs 1 page fault during each phase, the algorithm pays at least $\min\{H_k, \frac{1}{k \cdot \alpha}\}$, hence by summing over all phases of $\sigma$, we arrive at the desired competitive ratio. ◀

Next, we give an improved lower bound for lazy algorithms for paging. Recall that lazy algorithms for paging are the algorithms that are never allowed to change its cache configuration unless there is a page miss. This class includes RandomMark as well as most other known online paging algorithms. Note that this definition is slightly more general than the usual definition of lazy algorithms, where the algorithm is only allowed to fetch one page per request [19]; the intention of this definition is that the lower bound holds for metrical task systems as well. In the classic setting without infused advice, any algorithm can be turned to a lazy algorithm without increasing its cost; note, however, that the transformed algorithm may not be randomness-oblivious. If we restrict our attention to randomness-oblivious algorithms, the non-lazy algorithms may have an advantage over the lazy algorithms due to non-lazy algorithm's potentially frequent interaction with the oracle, which could be used by the oracle to give advice to prefetch some items even before the first cache miss occurs.

▶ **Theorem 5.4.** *Assume that an online randomized algorithm with RIA for online paging is lazy, randomness-oblivious and c-competitive against the oblivious adversary. Then* $c \geq \min\{H_k, \frac{1}{\alpha}\}$*, where* $\alpha$ *is the infusion parameter.*

**Proof.** To prove the theorem, we apply Yao's Minimax Principle [52] to competitiveness of randomized algorithms. Consider any deterministic advice-oblivious online algorithm and the probability distribution for choosing a request sequence as in the proof of Theorem 5.3.

We claim that for any advice-oblivious algorithm, the advice received in past phases cannot reduce the cost of the algorithm in future phases. We argue as follows. First, the advice-oblivious algorithm is forbidden to store past advice in its internal memory for future use. Second, no algorithm can store meaningful advice for the future in its cache configuration: each phase contains requests to $k+1$ different items, so for any cache configuration at the start of the phase, there is always at least 1 *clean page*: a page that is requested in the phase that the algorithm does not have in the cache at the start of the phase.

We will show that the expected cost of the algorithm is at least $\min\{H_k, \frac{1}{\alpha}\}$ in any phase. We lower-bound the cost of the algorithm in each phase in two ways, depending on whether in this phase the algorithm receives advice in some round with a cache miss or not.

1. Assume that the algorithm does not receive advice in any round with a cache miss. Since the algorithm is lazy, advice received in rounds without cache misses does not influence the algorithm's cache configuration, and since the algorithm is advice-oblivious, it cannot store such advice either. In such case, the algorithm acts as an online algorithm without advice throughout the phase, and the expected cost of the algorithm in the phase is at least $H_k$, following the standard arguments [44]: the expected length of the phase is $k \cdot H_k$, the average cost of the algorithm for each request is $1/k$ because the requested page is random and each of its pages is outside the cache with equal probability, therefore the cost of the algorithm within a phase is $H_k$.
2. Assume that the algorithm receives advice in a round with a cache miss. To receive advice at a round with a cache miss, we need in expectation $1/\alpha$ rounds with cache misses. Each round with a cache miss costs 1, hence the expected cost of the algorithm is at least $1/\alpha$.

An optimal offline algorithm OPT incurs a single page fault during each phase, and the algorithm pays at least $\min\{H_k, \frac{1}{\alpha}\}$, hence by summing over all phases of $\sigma$, we arrive at the desired competitive ratio. ◀

The bound given in Theorem 5.4 is asymptotically tight for lazy algorithms. However, a gap of a constant factor of 2 remains. To address this gap, an optimal randomized algorithm for paging [43] may be a possible direction for future studies.

## 6 Future Work

Our work opens interesting avenues for future research. In particular it will be interesting to further explore the utility of our method applied to other randomized online algorithms. Randomness-oblivious online algorithms are known for many online problems, e.g., all randomized memoryless algorithms [26] such as the COINFLIP algorithm for file migration [51] or the HARMONIC algorithm for $k$-server [14, 45] are randomness-oblivious.

## References

1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000.

2 Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. Syst. Sci.*, 70(2):145–175, 2005.

3 Susanne Albers and Dario Frascaria. Quantifying competitiveness in paging with locality of reference. *Algorithmica*, 80(12):3563–3596, 2018.

4 Susanne Albers and Maximilian Janke. Scheduling in the random-order model. *Algorithmica*, 83(9):2803–2832, 2021.

5 Susanne Albers, Arindam Khan, and Leon Ladewig. Best fit bin packing with random order revisited. *Algorithmica*, 83(9):2833–2858, 2021.

6 Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021.

7 Susanne Albers and Sonja Lauer. On list update with locality of reference. *J. Comput. Syst. Sci.*, 82(5):627–653, 2016.

8 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006.

9 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009.

10 Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020 (ITCS)*, volume 151, pages 52:1–52:15, 2020.

11 Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 2020.

12 Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

13 Nikhil Bansal, Christian Coester, Ravi Kumar, Manish Purohit, and Erik Vee. Learning-augmented weighted paging. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 67–89. SIAM, 2022.

14 Yair Bartal and Eddie Grove. The harmonic $k$-server algorithm is competitive. *J. ACM*, 47(1):1–15, 2000.

15 Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multi-level feedback algorithm. In *44th Symposium on Foundations of Computer Science (FOCS 2003)*, pages 462–471, 2003.

16 Laszlo A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Syst. J.*, 5(2):78–101, 1966.

17 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

18 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. Online algorithms with advice: The tape model. *Inf. Comput.*, 254:59–83, 2017.

19 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

20 Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *J. Comput. Syst. Sci.*, 50(2):244–258, 1995.

21 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

22 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2), 2017.

23 Joan Boyar, Sandy Irani, and Kim S. Larsen. A comparison of performance measures for online algorithms. *Algorithmica*, 72(4):969–994, 2015.

24 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263, 2009.

25 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

26 Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64. DIMACS/AMS, 1991.

27 José R. Correa, Andrés Cristi, Laurent Feuilloley, Tim Oosterwijk, and Alexandros Tsigonias-Dimitriadis. The secretary problem with independent sampling. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2047–2058. SIAM, 2021.

28 Stefan Dobrev, Jeff Edmonds, Dennis Komm, Rastislav Královic, Richard Královic, Sacha Krug, and Tobias Mömke. Improved analysis of the online set cover problem with advice. *Theor. Comput. Sci.*, 689:96–107, 2017.

29 Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, 2005.

30 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011.

31 Yuval Emek, Yuval Gil, Maciej Pacut, and Stefan Schmid. Online algorithms with randomly infused advice. *CoRR*, abs/2302.05366, 2023. doi:10.48550/arXiv.2302.05366.

32 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

33 Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

34 Greg N. Frederickson. Probabilistic analysis for simple one- and two-dimensional bin packing algorithms. *Inf. Process. Lett.*, 11:156–161, 1980.

35 Edward F. Grove. Online bin packing with lookahead. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–436. ACM/SIAM, 1995.

36 Sandy Irani and Steven S. Seiden. Randomized algorithms for metrical task systems. *Theor. Comput. Sci.*, 194(1-2):163–182, 1998.

37 Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

38 Anna R. Karlin and Elias Koutsoupias. Beyond competitive analysis. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 529–546. Cambridge University Press, 2020. doi:10.1017/9781108637435.031.

39 Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. Randomized online algorithms with high probability guarantees. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 25, pages 470–481, 2014.

40 Simon Korman. On the use of randomization in the online set cover problem. *Master's thesis, Weizmann Institute of Science, Rehovot, Israel*, 2004.

41 Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30(1):300–317, 2000.

42 Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021.

43 Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6:816–825, 1991.

44 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

45 Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC Press, 1999.

**46** Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pages 9684–9693, 2018.

**47** Jan Reineke and Alejandro Salinger. On the smoothness of paging algorithms. *Theory Comput. Syst.*, 62(2):366–418, 2018.

**48** Ronald L. Rivest. On self-organizing sequential search heuristics. *Commun. ACM*, 19(2):63–67, 1976.

**49** Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020.

**50** Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

**51** Jeffery R. Westbrook. Randomized algorithms for multiprocessor page migration. *SIAM J. Comput.*, 23(5):951–965, 1994.

**52** Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science, Providence*, pages 222–227. IEEE Computer Society, 1977.

**53** Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994. doi:10.1007/BF01189992.

# The Lawn Mowing Problem: From Algebra to Algorithms

**Sándor P. Fekete**
Department of Computer Science, TU Braunschweig, Germany

**Dominik Krupke**
Department of Computer Science, TU Braunschweig, Germany

**Michael Perk**
Department of Computer Science, TU Braunschweig, Germany

**Christian Rieck**
Department of Computer Science, TU Braunschweig, Germany

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

## Abstract

For a given polygonal region $P$, the Lawn Mowing Problem (LMP) asks for a shortest tour $T$ that gets within Euclidean distance $1/2$ of every point in $P$; this is equivalent to computing a shortest tour for a unit-diameter cutter $C$ that covers all of $P$. As a generalization of the Traveling Salesman Problem, the LMP is NP-hard; unlike the discrete TSP, however, the LMP has defied efforts to achieve exact solutions, due to its combination of combinatorial complexity with continuous geometry.

We provide a number of new contributions that provide insights into the involved difficulties, as well as positive results that enable both theoretical and practical progress. (1) We show that the LMP is algebraically hard: it is not solvable by radicals over the field of rationals, even for the simple case in which $P$ is a $2 \times 2$ square. This implies that it is impossible to compute exact optimal solutions under models of computation that rely on elementary arithmetic operations and the extraction of $k$th roots, and explains the perceived practical difficulty. (2) We exploit this algebraic analysis for the natural class of polygons with axis-parallel edges and integer vertices (i.e., polyominoes), highlighting the relevance of turn-cost minimization for Lawn Mowing tours, and leading to a general construction method for feasible tours. (3) We show that this construction method achieves theoretical worst-case guarantees that improve previous approximation factors for polyominoes. (4) We demonstrate the practical usefulness *beyond polyominoes* by performing an extensive practical study on a spectrum of more general benchmark polygons: We obtain solutions that are better than the previous best values by Fekete et al., for instance sizes up to 20 times larger.

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

**Keywords and phrases** Geometric optimization, covering problems, tour problems, lawn mowing, algebraic hardness, approximation algorithms, algorithm engineering

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.45

**Related Version** *Full Version*: `https://arxiv.org/abs/2307.01092`

**Supplementary Material** *Software (Source Code)*: `https://github.com/tubs-alg/lawn-mowing-from-algebra-to-algorithms`, archived at `swh:1:dir:2e2d891d6ee3b0efcf1324f0b123118b93d8ba69`

**Funding** Work at TU Braunschweig has been partially supported by the German Research Foundation (DFG), project "Computational Geometry: Solving Hard Optimization Problems" (CG:SHOP), grant FE407/21-1.





# 1 Introduction

Many geometric optimization problems are NP-hard: the number of possible solutions is finite, but there may not be an efficient method for systematically finding a best one. A different kind of difficulty considered in geometry is rooted in the impossibility of obtaining solutions with a given set of construction tools: Computing the length of a diagonal of a square is not possible with only rational numbers; trisecting any given angle cannot be done with ruler and compass, and neither can a square be constructed whose area is equal to that of a given circle.

In this paper, we consider the *Lawn Mowing Problem* (LMP), in which we are given a polygonal region $P$ and a disk cutter $C$ of diameter 1; the task is to find a closed roundtrip of minimum Euclidean length, such that the cutter "mows" all of $P$ , i.e., a shortest tour that moves the center of $C$ within distance $1/2$ from every point in $P$. The LMP naturally occurs in a wide spectrum of practical applications, such as robotics, manufacturing, farming, quality control, and image processing, so it is of both theoretical and practical importance. As a generalization of the classic Traveling Salesman Problem (TSP), the LMP is also NP-hard; however, while the TSP has shown to be amenable to exact methods for computing provably optimal solutions even for large instances [1], the LMP has defied such attempts, with only recently some first practical progress by Fekete et al. [26].

## 1.1 Related Work

There is a wide range of practical applications for the LMP, including manufacturing [5, 31, 32], cleaning [12], robotic coverage [13, 15, 29, 35], inspection [21], CAD [20], farming [6, 16, 40], and pest control [9]. In Computational Geometry, the Lawn Mowing Problem was first introduced by Arkin et al. [3], who later gave the currently best approximation algorithm with a performance guarantee of $2\sqrt{3}\alpha_{\mathrm{TSP}} \approx 3.46\alpha_{\mathrm{TSP}}$ [4], where $\alpha_{\mathrm{TSP}}$ is the performance guarantee for an approximation algorithm for the TSP.

Optimally covering even relatively simple regions such as a disk by a set of $n$ *stationary* unit disks has received considerable attention, but is excruciatingly difficult; see [10, 11, 28, 33, 36, 38]. As recently as 2005, Fejes Tóth [22] established optimal values for the maximum radius of a disk that can be covered by $n = 8, 9, 10$ unit circles. Recent progress on covering by (not necessarily equal) disks has been achieved by Fekete et al. [23, 24].

A first *practical* breakthrough on computing provably good Lawn Mowing tours was achieved by Fekete et al. [26], who established a primal-dual algorithm for the LMP by iteratively covering an expanding *witness set* of finitely many points in $P$. In each iteration, their method computes a lower bound, which involves solving a special case of a TSP instance with neighborhoods, the *Close-Enough TSP* (CETSP) to provable optimality; for an upper bound, the method is enhanced to provide full coverage. In each iteration, this establishes both a valid solution and a valid lower bound, and thereby a bound on the remaining optimality gap. They also provided a computational study, with good solutions for a large spectrum of benchmark instances with up to 2000 vertices. However, this approach encounters scalability issues for larger instances, due to the considerable number of witnesses that need to be placed.

A seminal result on algebraic aspects of geometric optimization problems was achieved by Bajaj [7], who established algebraic hardness for the Fermat-Weber problem of finding a point in $\mathbb{R}^2$ that minimizes the sum of Euclidean distances to all points in a given set. Others have studied the Galois complexity for geometric problems like Graph Drawing or the Weighted Shortest Path Problem [8, 14, 39].

As we will see in the course of our algorithmic analysis the number of turns in a tour is of crucial importance for the overall cost; this has been previously studied by Arkin et al. [2] in a discrete setting. This objective is also of practical importance in the context of physical coverage, e.g., in the context of efficient drone trajectories [9].

## 1.2 Our Results

We provide a spectrum of new theoretical and practical results for the Lawn Mowing Problem.

- We prove that computing an optimal Lawn Mowing tour is algebraically hard, even for the case of mowing a $2 \times 2$ square by a unit-diameter disk, as it requires computing zeroes of high-order irreducible polynomials.
- We exploit the algebraic analysis to achieve provably good trajectories for polyominoes, based on the consideration of turn cost, and provide a method for general polygons.
- We show that this construction method achieves theoretical worst-case guarantees that improve previous approximation factors for polyominoes.
- We demonstrate the practical usefulness *beyond polyominoes* on a spectrum of more general benchmark polygons, obtaining better solutions than the previous values by Fekete et al. [26], for instance sizes up to 20 times larger.

## 1.3 Definitions

A (simple) *polygon* $P$ is a (non-self-intersecting) shape in the plane, bounded by a finite number $n$ of line segments. The *boundary* of a polygon $P$ is denoted by $\partial P$. A *polyomino* is a polygon with axis-parallel edges and vertices with integer coordinates; any polyomino can be canonically partitioned into a finite number $N$ of unit-squares, called *pixels*. A *tour* is a closed continuous curve $T : [0, 1] \to \mathbb{R}^2$ with $T(0) = T(1)$. The *cutter* $C$ is a disk of diameter $d$, centered in its midpoint. Without loss of generality, we assume $d = 1$ for the rest of the paper. The *coverage* of a tour $T$ with the disk cutter $C$ is the Minkowski sum $T \oplus C$. A *Lawn Mowing tour* $T$ of a polygon $P$ with a cutter $C$ is a tour whose coverage contains $P$. An *optimal* Lawn Mowing tour is a Lawn Mowing tour of shortest length.

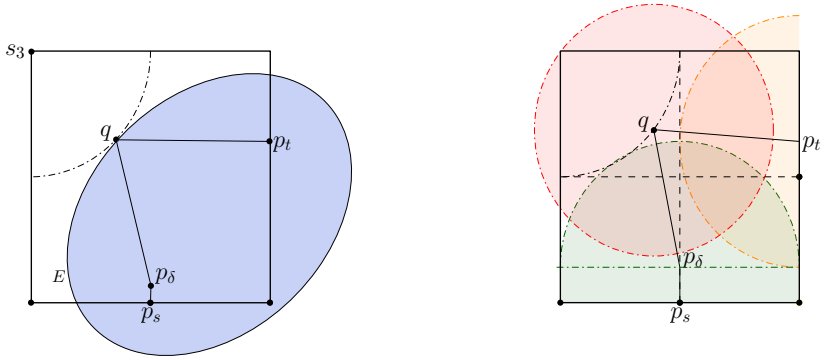
# 2 Algebraic Hardness

In their recent work, Fekete et al. [26] prove that an optimal Lawn Mowing tour for a polygonal region is necessarily polygonal itself; on the other hand, they show that optimal tours may need to contain vertices with irrational coordinates corresponding to arbitrary square roots, even if $P$ is just a triangle. In the following, we show that if $P$ is a $2 \times 2$ square, an optimal tour may involve coordinates that cannot even be described with radicals. See Figure 2 for the structure of optimal trajectories.

▶ **Theorem 1.** *For the case in which $P$ is a $2 \times 2$ square, the Lawn Mowing Problem is algebraically hard: an optimal tour involves coordinates that are zeroes of polynomials that cannot be expressed by radicals.*

## 2.1 Optimal Tours at Corners

For the $2 \times 2$ square $P$, consider the upper left $1 \times 1$ subsquare $S_0$ with corners $(0, 0)$, $(0, 1)$, $(1, 1)$, $(0, 1)$, further subdivided into four $1/2 \times 1/2$ quadrants $S_{0,0}, \ldots, S_{0,3}$, and an optimal path $\omega$ that enters $S_0$ at the bottom and leaves it to the right. Let $p_s = (p_s^x, 0)$, $p_t = (1, p_t^y)$




**(a)** Any $0 \le \delta \le 1$ defines $p_\delta, p_t, q$ and ellipse $E$. **(b)** The optimal path $\omega$ through $S_0$.

**Figure 1** Visualizations for Lemma 4.

be the points where $\omega$ enters and leaves $S_0$, respectively. For the following lemmas, we assume that a covering path exists that obeys the above conditions. We will later determine that path and show that it covers $S_0$. The proofs of Lemmas 2, 3, and 5 can be found in the full version [27].

▶ **Lemma 2.** *$p_s^x \le 1/2$ and $p_t^y \ge 1/2$ and either $p_s^x = 1/2$ or $p_t^y = 1/2$.*

Without loss of generality, we assume that $p_s^x = 1/2$. The next step is to find the optimal position of $p_t$. As an optimal path $\omega$ must enter the quadrant $S_{0,3}$ once, we can subdivide the path into two parts. For some $\delta > 0$, let $p_t^y = 1/2 + \delta$ and $p_\delta = (1/2, \delta)$.

▶ **Lemma 3.** *For any $\delta > 0$, $\omega$ has a subpath $p_s p_\delta$.*

▶ **Lemma 4.** *The uniquely-shaped optimal Lawn Mowing path $\omega$ between two adjacent sides of $S_0$ has length $L_{S_0} \approx 1.309$ with $\omega = (p_s, p_\delta, q, p_t)$ and*

$$p_s = (1/2, 0) \quad p_\delta = (1/2, \delta) \approx (1/2, 0.168) \quad q \approx (0.386, 0.682) \quad p_t = (1, 1/2+\delta) \approx (1, 0.668).$$

**Proof.** Let $s_3$ be the top left corner of $S_0$. We identify a shortest path for visiting one point $q$ on a circle $U$ with diameter 1 centered in $s_3$ dependent on $\delta$, a necessary condition for a feasible path. Let $c = d(p_\delta, q) + d(q, p_t)$ be the distance from both points to $U$. Consider an ellipse $E$ with foci $p_\delta, p_t$ that touches $U$ in a single point, see Figure 1a. By definition, the intersection point $q$ minimizes the distance $c$. For $\delta \in [0, 1]$ we want to find an intersection point between $E$ and $U$ that minimizes distance $c$. We can solve this problem with an exact optimization approach using Mathematica, see the full version of our paper [27]. It turns out that $\delta, q^x, q^y$ can only be expressed as the first, third, and first roots of three irreducible high-degree polynomials $f_\delta, f_{q^x}, f_{q^y}$, see Equation (1) and the full version [27].

$$\begin{aligned} f_\delta(x) =& 589\,824x^{16} - 7\,077\,888x^{15} + 41\,189\,376x^{14} - 154\,386\,432x^{13} + \\ & 416\,788\,480x^{12} - 857\,112\,576x^{11} + 1\,383\,417\,856x^{10} - 1\,779\,354\,624x^9 + \\ & 1\,834\,437\,632x^8 - 1\,514\,108\,928x^7 + 992\,782\,336x^6 - 509\,312\,064x^5 + \\ & 199\,354\,208x^4 - 57\,160\,752x^3 + 11\,200\,088x^2 - 1\,313\,928x + 67\,417 \end{aligned} \tag{1}$$

The value for $\delta \approx 0.167876$ defines the points $p_\delta$ and $p_t$. Together with the values for $q^x, q^y$, we obtain the path above. The combined length of the path is $\delta + c \approx 1.308838224$. As $\omega$ contains a subpath that crosses the full height of $S_{0,0}$ and another subpath that crosses the full width

of $S_{0,2}$, both quadrants are covered by $\omega$, see Figure 1b. By construction, the bottom right quadrant is covered by the segment $p_s p_\delta$ and the point $p_t$. The top left quadrant is covered by $q$, because $S_{0,3}$ is fully contained in a disk with diameter 1 centered in $q$. Therefore, $\omega$ is a feasible path between two adjacent edges of $S_0$ with a length of $L \approx 1.309$. ◀

▶ **Lemma 5.** *A square $P$ of side length 2 has a uniquely-shaped optimal Lawn Mowing tour $T$ of length $L = 4L_{S_0}$, where $L_{S_0} \approx 1.309$.*

## 2.2 Galois Group of the Polynomial

Now we show that the coordinates of the optimal path $\omega$ can not be expressed by radicals. We employ a similar technique as Bajaj [7] for the generalized Weber problem. A *field* $K$ is said to be an *extension* (written as $K/\mathbb{Q}$) of $\mathbb{Q}$ if $K$ contains $\mathbb{Q}$. Given a polynomial $f(x) \in \mathbb{Q}[x]$, a finite extension $K$ of $\mathbb{Q}$ is a *splitting field* over $\mathbb{Q}$ for $f(x)$ if it can be factorized into linear polynomials $f(x) = (x - a_1) \cdots (x - a_k) \in K[x]$ but not over any proper subfield of $K$. Alternatively, $K$ is a splitting field of $f(x)$ of degree $n$ over $\mathbb{Q}$ if $K$ is a minimal extension of $\mathbb{Q}$ in which $f(x)$ has $n$ roots. Then the *Galois group* of the polynomial $f$ is defined as the Galois group of $K/\mathbb{Q}$. In principle, the Galois group is a certain permutation group of the roots of the polynomial. From the fundamental theorem of Galois theory, one can derive a condition for solvability by radicals of the roots of $f(x)$ in terms of algebraic properties of its Galois group. We state three additional theorems from Galois theory and Bajaj's work. The proofs can be found in [7, 34].

▶ **Lemma 6** ([34])**.** *$f(x) \in \mathbb{Q}[x]$ is solvable by radicals over $\mathbb{Q}$ iff the Galois group over $\mathbb{Q}$ of $f(x)$, $Gal(f(x))$, is a solvable group.*

▶ **Lemma 7** ([34])**.** *The symmetric group $S_n$ is not solvable for $n \geq 5$.*

▶ **Lemma 8** ([7])**.** *If $n \equiv 0 \mod 2$ and $n > 2$ then the occurrence of an $(n-1)$-cycle, an $n$-cycle, and a permutation of type $2 + (n-3)$ on factoring the polynomial $f(x)$ modulo primes that do not divide the discriminant of $f(x)$ establishes that $Gal(f(x))$ over $\mathbb{Q}$ is the symmetric group $S_n$.*

**Proof of Theorem 1.** It suffices to show that $f_\delta$ is not solvable by the radicals as it describes the y-coordinates of two points in the solution. We provide three factorizations of $f_\delta$ modulo three primes that do not divide the discriminant $disc(f_\delta(x))$.

$$\begin{aligned} f_\delta(x) \equiv\ & 12(x^{16} + 11x^{15} + 20x^{14} + 20x^{13} + 12x^{12} + 15x^{11} + 20x^{10} + 22x^9 + 19x^8 + 2x^7 + \\ & 18x^6 + 10x^5 + 12x^4 + 19x^3 + 16x^2 + 9x + 8) \mod 23 \\ f_\delta(x) \equiv\ & 21(x + 44)(x^2 + 34x + 39)(x^{13} + 4x^{12} + x^{11} + 41x^{10} + 12x^9 + 21x^8 + 24x^7 + \\ & 32x^5 + 22x^4 + 10x^3 + 24x^2 + 18x + 13) \mod 47 \\ f_\delta(x) \equiv\ & (x + 39)(x^{15} + 8x^{14} + 43x^{13} + 23x^{12} + 19x^{11} + 38x^{10} + 9x^9 + 6x^8 + 17x^7 + \\ & 34x^6 + 46x^5 + 43x^4 + 27x^3 + 50x^2 + 56x + 1) \mod 59 \end{aligned}$$

For the good primes $p = 23, 47$, and 59, the degrees of the irreducible factors of $f_\delta(x)$ mod $p$ gives us an $16 - cycle$, a $2 + 13$ permutation and a 15-cycle, which is enough to show with Lemma 8 and $n = 16$ that $Gal(f_\delta) = S_{16}$. By Lemma 7, $S_{16}$ is not solvable; with Lemma 6, this proves the theorem. ◀

# 3 Mowing Polyominoes

In the following, we analyze good tours for *polyominoes*, which naturally arise when a geometric (or geographic) region is mapped, resulting in axis-parallel edges and integer vertices. In the subsequent two sections, we describe the ensuing theoretical worst-case guarantees (Section 4) and the practical performance (Section 5).

## 3.1 Combinatorial Bounds

For a unit-square cutter, the LMP on polyominoes naturally turns into the TSP on the dual grid graph induced by pixel centers.

▶ **Lemma 9.** *Let $N \geq 2$ be the area of a polyomino $P$ to be mowed with a unit-square cutter, and let $L$ be the minimum length of a Lawn Mowing tour. Then $L \geq N$. In the case of a unit-square cutter, $L = N$ iff the dual grid graph of $P$ has a Hamiltonian cycle.*

This follows from Lemma 2 in the paper by Arkin et al. [4] (which argues that there is an optimal LMP tour for a polyomino whose vertices are pixel centers) and implies the NP-hardness of the LMP (Theorem 1 in [4]). In particular, they focused on grid graphs without a *cut vertex*, which is a node $v$ whose removal disconnects $G$: "If $G$ has a cut vertex $v$, then we can consider separately the approximation problem in each of the components obtained by removing $v$, and then splice the tours back together at the vertex $v$ to obtain a tour in the entire graph $G$. Thus, we concentrate on the case in which $G$ has no cut vertices."

For a simply connected polyomino consisting of $N$ pixels, the corresponding grid graph $G$ does not have any *holes*, i.e., the complement of $G$ in the infinite integer lattice is connected. These allow a tight combinatorial bound on the tour length. If $G$ has no cut vertices, then a combinatorially bounded tour of $G$ exists, as noted by Arkin et al. [4] as follows.

▶ **Theorem 10** (Theorem 5 in [4]). *Let $G$ be a simple grid graph, having $N$ nodes at the centerpoints, $V$, of pixels within a simple rectilinear polygon, $R$, having $n$ (integer-coordinate) sides. Assume that $G$ has no cut vertices. Then, in time $O(n)$, one can find a representation of a tour, $T$, that visits all $N$ nodes of $G$, of length at most $\frac{6N-4}{5}$.*

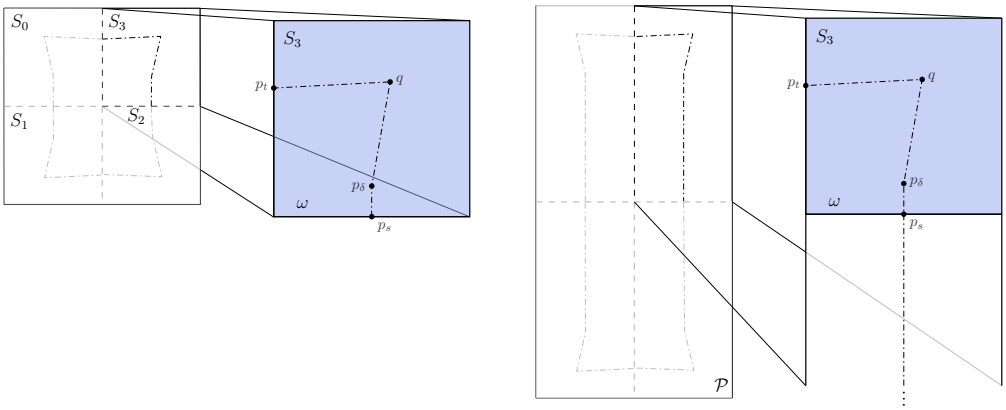
For polyominoes with holes, there is a slightly worse, but still relatively tight combinatorial bound of $\frac{53N}{40} = 1.325N$ for the tour length, as follows.

▶ **Theorem 11** (Theorem 7 in [4]). *Let $G$ be a connected grid graph, having $N$ nodes at the centerpoints, $V$, of pixels within a (multiply connected) rectilinear polygon, $R$, having $n$ (integer-coordinate) sides. Assume that $G$ has no local cut vertices. Then, in time $O(n)$, one can find a representation of a tour, $T$, that visits all $N$ nodes of $G$, of length at most $1.325N$.*

## 3.2 Mowing with a Disk

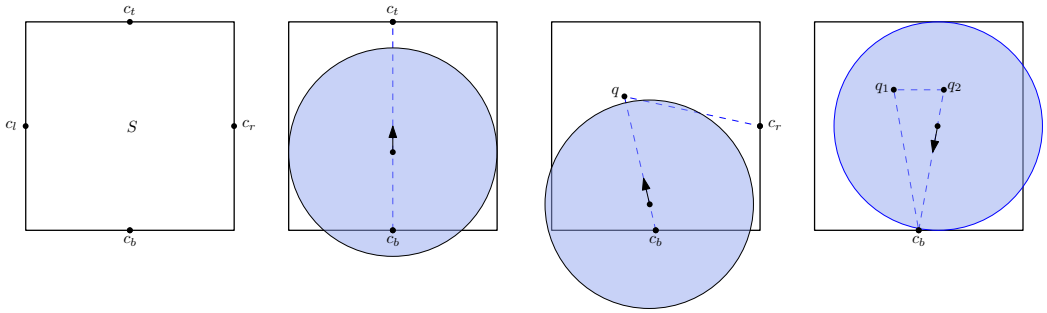
The natural lower bound of Lemma 9 still applies when mowing with a circular cutter, because any unit distance covered by the cutter can at most cover a unit area. However, meeting (or approximating) this bound is no longer possible by simply finding a Hamiltonian cycle (or a good tour) in the underlying grid graph, as a circular cutter may cover already mowed area or area outside of $P$ when dealing with pixel corners. Minimizing this effect ultimately leads to the algebraic analysis from the previous section.

A starting point for further insights is illustrated in Figure 2: The optimal path from Lemma 4 with length $L_{S_0}$ can be used for rectangles with width 2 and arbitrary height $h \geq 2$.




**(a)** Optimal Lawn Mowing tour for a $2 \times 2$ square. **(b)** Optimal Lawn Mowing tour for a rectangle.

**Figure 2** Optimal Lawn Mowing tours for a square and a rectangle.




**(a)** A pixel $S$ with transition points. **(b)** A straight-line covering, with zero turn. **(c)** A covering for a single, 90-degree turn. **(d)** A covering for a double, 180-degree U-turn.

**Figure 3** A pixel $S$ with three elementary covering trajectories.

▶ **Corollary 12.** *Any rectangle $P$ with width 2 and height $h > 2$ has a uniquely-shaped optimal Lawn Mowing tour $T$ of length $L = 4L_{S_0} + 2h - 8$.*

Extending this idea to more general polyominoes leads to realizing a tour of the dual grid with locally optimal "puzzle pieces": a limited set of locally good trajectories that mow each visited pixel, which are merged at transition points on the pixel boundaries; see Figure 3a. The construction of the puzzle pieces is done in Section 3.3.

## 3.3 Constructing Puzzle Pieces

In order to analyze locally good trajectories for mowing visited pixels, consider the four corners of a pixel with coordinates $(0,0), (1,0), (1,1), (0,1)$. We consider *transition points* $c_b = (1/2, 0)$, $c_r = (1, 1/2)$, $c_t = (1/2, 1)$, and $c_l = (0, 1/2)$ at the edge centers to ensure an overall connected trajectory, as shown in Figure 3a. There are three combinatorially distinct ways for visiting a pixel, corresponding to Figures 3b–3d. These are (i) a straight path with length 1, (ii) a simple turn with length $\approx 1.32566$, and (iii) a U-turn with length $\approx 1.611183$, see the full version [27] for details on how to obtain these paths. Note that we do not use the optimal path from Lemma 4, because it uses transition points that are slightly off center, $p_t \neq c_r$, with the imbalance canceled out between two adjacent simple turns. Thus, using central transition points incurs a small marginal cost when compared to an optimal trajectory (1.32566 vs. 1.309, or about 1.2% longer for each simple turn), but it sidesteps the higher-order difficulties of combining longer off-center strips.

### 3.4 Building an Overall Tour

Making use of the puzzle pieces, we can now approach the LMP in three steps, as follows.

**A** Find a cheap roundtrip on the dual grid graph.
**B** Carry out the individual pixel transitions based on the above puzzle pieces as building blocks to ensure coverage of all pixels and thus a feasible tour.
**C** Perform post-processing sensitive to the transition costs on the resulting tour to achieve further improvement.

In the following sections, we describe how the involved steps can be carried out either with an emphasis on worst-case runtime and worst-case performance guarantee (giving rise to theoretical approximation algorithms, as discussed in the following Section 4), or with the goal of good practical performance in reasonable time for a suite of benchmark instances (leading to the experimental study described in Section 5).

## 4 Theoretical Performance: Approximation

For constant-factor approximation, we start with a low-cost roundtrip in the dual grid graph (Step A), e.g., with the previous results of Arkin et al. [4]. Step B is realized using the puzzle pieces of Section 3.3 for a feasible tour, at a cost of $1 + \tau := 1.32566$ for each 90-degree turn in the grid tour (corresponding to piece (ii)); note that the turn cost for a U-turn of 1.61118 (corresponding to piece (iii)) does not exceed $1 + 2\tau$. By using combinatorial arguments for the post-processing Step C, we can prove that a limited number of covering turns (with an additional turn cost $\tau$) suffices for overall feasibility.

▶ **Theorem 13.** *Let $P$ be a polyomino with $N > 5$ pixels, and let $T$ be a tour of the dual grid graph of length $L$. Then we can find a feasible Lawn Mowing tour for a unit-diameter disk of length at most $L(1 + \tau)$.*

**Proof.** Let $T$ be a tour of the dual grid graph; let $L$ be the length of $T$. $L$ is the total number of visits of individual pixels, inducing the following three categories of pixel visits.

1. $L_0$ "free" visits of pixels, in which no covering turn occurs, and no turn cost is incurred.
2. $L_1$ "one-turn" visits of pixels, in which one covering turn occurs, for a turn cost of $\tau$.
3. $L_2$ "U-turn" visits of pixels, in which a double covering turn occurs, for a turn cost of not more than $2\tau$.

Let $p_i$ be a pixel that is visited in step $i$ of the tour by a U-turn of $T$. Then $p_i$ is adjacent to a pixel $q = p_{i-1} = p_{i+1}$ that was left in step $i$ and entered in step $i + 1$. Because no pixel visited by a U-turn needs to be visited more than once, as well as $N > 5$, the pixel $q$ cannot only have neighbors that are visited by U-turns. Therefore, $q$ has a predecessor in the tour that is not a U-turn, (w.l.o.g., $p_{i-2}$); this visit from $p_{i-2}$ is either a one-turn visit with a covering turn, or a free visit. In either case, $q$ is already covered when visited from $p_i$, and we can simply follow the grid path at only the distance cost of 1.

As a consequence, each U-turn visit (incurring a cost not exceeding $2\tau$) can be uniquely mapped to a free visit of its successor (incurring no turn cost), and the overall cost for all covering turns does not exceed $L\tau$, for a total length of at most $L(1 + \tau)$, as claimed. ◀

For simple polyominoes without cut vertices, Theorem 10 provides a tour $T$ in the dual grid graph of length at most $\frac{6N-4}{5}$, implying the following.

▶ **Corollary 14.** *Let $P$ be a simple polyomino with $n$ vertices and $N$ pixels, whose dual grid graph does not have any cut vertices. Then, in time $O(n)$, one can find a representation of a feasible Lawn Mowing trajectory $T$ for a unit-diameter disk of length at most $\frac{6N-4}{5}\tau$, which is within* 1.5908 *of the optimum.*

For polyominoes with holes, we can apply the same line of argument to a tour $T$ of the dual grid graph obtained from Theorem 11.

▶ **Corollary 15.** *Let $P$ be a (not necessarily simple) polyomino with $n$ vertices and $N$ pixels, whose dual grid graph does not have any cut vertices. Then, in time $O(n)$, one can find a representation of a feasible Lawn Mowing trajectory $T$ for a unit-diameter disk of length at most $\frac{53N}{40}\tau$, which is within* 1.7565 *of the optimum.*

As the number of turns is of critical importance for the overall cost of a Lawn Mowing tour obtained from a tour of the dual grid graph, we can consider optimizing a linear combination of tour length and turn cost. Arkin et al. [2] gave a PTAS for this problem, as follows.

▶ **Theorem 16** (Theorem 5.17 in [2])**.** *Define the cost of a tour to be its length plus $C$ times the number of (90-degree) turns. For any fixed $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm, with running time $2^{O(h)}N^{O(C)}$, for minimizing the cost of a tour for an integral orthogonal polygon $P$ with $h$ holes and $N$ pixels.*

Combining tour length and turns allows providing more explicit bounds, as follows. Additional local considerations are possible, but these do not necessarily improve the worst-case bounds. Instead, they are employed heuristically in the practical section.

▶ **Theorem 17.** *Let $P$ be a polyomino with $n$ vertices and $N$ pixels, and let $T$ be a tour of the dual grid graph of length $L$ and a total of $t$ (weighted) turns. Then there is a feasible Lawn Mowing tour of cost at most $L + t\tau$.*

# 5 Practical Performance: Algorithm Engineering

## 5.1 Algorithmic Tools

Here we exploit the algorithmic approach of Section 3.4 for good *practical* performance for general polygonal regions, starting with a preprocessing step: For a given polygonal region $Q$, find a suitable polyomino $P$ that covers it.

We can then aim for practical minimization of tour length and turn cost for **A** (analogous to the theoretical Theorem 16), and use puzzle pieces in **B** for a feasible tour. In principle, we can approach **A** by considering an integer program (IP); however, solving this IP becomes too costly for larger instances, so we use a more scalable approach: **(A)** Find a good TSP solution on the dual grid graph; **(B)** insert puzzle pieces; **(C)** minimize the induced turn cost by Integer Programming and Large Neighborhood Search (LNS).

### 5.1.1 Choosing a Suitable Grid

Consider a non-degenerate polygonal region $Q$, and a minimal covering polyomino $P$ of cell size $\ell$. Without loss of generality, $Q$ contains only pixels with a point of $Q$ in their interior; furthermore, we can assume that both an $x$- and a $y$-coordinate of a grid point coincide with a coordinate of $Q$. This limits the number of relevant grid positions to a quadratic number of choices, from which one can choose the one with the smallest number of pixels contained in the resulting polygon $P$.

### 5.1.2 Minimizing Tour and Turn Cost

Finding a covering tour of minimum combined tour length and turn cost can be formulated as an IP. As the cost for each turn can be specified individually in this IP, we can also minimize the final tour length directly instead of just approximating it based on the number of turns. In principle, this IP can be solved with CPLEX [17] or Gurobi [30]; however, this fails when aiming for truly large instances. (Even without the length of the tour, the turn-cost problem is notoriously difficult [25].) Thus, we have pursued an alternative approach that starts with a cheap roundtrip on the dual grid graph in which we ignore the turn cost. We then use this IP as part of a Large Neighborhood Search (described in Section 5.1.4) to minimize the actual costs of this solution, and for computing lower bounds on the best possible solution based on puzzle pieces.

**Formulating the Integer Program.** To formulate the integer program, let $\widehat{uvw}$ with $uv, vw \in E_H$ be the puzzle piece covering the pixel $v$ and connecting $c_{uv}$ and $c_{vw}$, and $\overline{uvw}$ be the direct path between $c_{uv}$ and $c_{vw}$. We call these tour elements (*covering* and *non-covering*) *tiles*. We use the variables $x_{\widehat{uvw}} \in \mathbb{B}, uv, vw \in E_H$ to denote which covering tile, i.e., puzzle piece, is used for $v \in V_P$ is in the tour. For simplicity, $x_{\widehat{uvw}}$ is also defined for $v \in V \setminus V_P$, but fixed to 0. Analogously, we are using the variables $x_{\overline{uvw}} \in \mathbb{N}_0, uv, vw \in E_H$ to denote how often which non-covering tiles, i.e., direct paths, for $v \in V$ are used in the tour. Because we may need to pass a pixel multiple times, this is an integer variable.

Finding the shortest set of cycles that cover all pixels $t \in V_P$ can be expressed as follows. Enforcing a single cycle, i.e., tour, is done later by some more complex constraints that need additional discussion.

$$\min \sum_{uv,vw \in E_H} ||\overline{uvw}|| \cdot x_{\overline{uvw}} + ||\widehat{uvw}|| \cdot x_{\widehat{uvw}} \tag{2}$$

$$\text{s.t.} \sum_{u,w \in N(v)} x_{\widehat{uvw}} = 1 \qquad \forall v \in V_P \tag{3}$$

$$\begin{aligned} &2 \cdot (x_{\overline{wvw}} + x_{\widehat{wvw}}) + \sum_{n \in N(v), n \neq w} (x_{\overline{nvw}} + x_{\widehat{nvw}}) \\ &= 2 \cdot (x_{\overline{vwv}} + x_{\widehat{vwv}}) + \sum_{n \in N(w), n \neq v} (x_{\overline{vwn}} + x_{\widehat{vwn}}) \end{aligned} \qquad \forall vw \in E_H \tag{4}$$

$$x_{\widehat{vwv}} \in \mathbb{B}, x_{\overline{vwv}} \in \mathbb{N}_0 \qquad \forall uv, vw \in E_H \tag{5}$$

The objective function (2) minimizes the sum of lengths of the used tiles (the length of a tile is denoted by $||\cdot||$). Equation (3) enforces that every pixel $v \in V_P$ that intersects the polygon $P$ has one covering tile; $N(v)$ are the neighbors of $v$. Equation (4) ensures that every tile has a matching incident tile on each end, i.e., connecting all tiles yields feasible cycles.

**Subtour Elimination.** Next, we have to add constraints that enforce a single tour. A simple, but insufficient, constraint is similar to the classical subtour elimination constraint of the Dantzig-Fulkerson-Johnson formulation [18] for the Traveling Salesman Problem. For every non-empty subset $S \subset V, S \neq \emptyset, V \not\subset S, V \setminus S \neq \emptyset$ that contains a real part of $V_P$, there has to be some path leaving the set to connect to $V \setminus S$.

$$\sum_{uv,vw \in E_H, v \in S, w \notin S} x_{\overline{uvw}} + x_{\widehat{uvw}} \geq 1$$

Unfortunately, this is not sufficient as we can have cycles that cross but are not connected, e.g., for the tiles $\overline{uvw}$ and $\widehat{svt}$ with $\{u, w\} \cap \{s, t\} = \emptyset$. While they share the same pixel $v$ in the grid graph, the paths themselves do not have to intersect. We can also not expect them to be exchangeable as this may increase the objective. Let $O$ be a cycle of tiles that cover only a real subset of $V_P$, $E(O)$ denote the edges in the grid graph, and $\widehat{abc} \in O$ be a covering tile of $O$ with $b \in V(O) \cap V_P$. The following constraint now forces the path that covers $v$ to change and connect to exterior parts.

$$\sum_{u,w \in N(b), \widehat{ubw} \notin O} x_{\widehat{ubw}} + \sum_{vw \in E(O), u \in N(v), \overline{uvw} \notin O, \widehat{uvw} \notin O} (x_{\overline{uvw}} + x_{\widehat{uvw}}) \geq 1$$

This constraint is sufficient as it can be applied to any cycle that is covering only a subset of $V_P$, but generally less efficient.

### 5.1.3 Finding a Cheap Roundtrip and Ensure Coverage

We consider two different methods for computing different initial tours.

**$\mathsf{TSP}_{\mathsf{Small}}$.** Previous authors [12, 37, 42, 44] have suggested using a grid graph $H'$ with smaller cell size $\ell = \frac{\sqrt{2}}{2}$ for covering $P$, or simply assumed square-shaped tools. This eliminates the need to consider any turn cost, as smaller pixels are covered when the cutter visits their centers. This yields $\mathrm{TSP}_{\mathrm{Small}}$, which we use as a baseline. Because of the smaller grid size, this may result in double coverage when parallel unit strips suffice to cover the $P$, for a worst-case overhead of $\sqrt{2} - 1$, or about 41.4 %.

**$\mathsf{TSP}_{\mathsf{Cov}}$.** As described in the preceding Section 3, we can use a cheap tour for the grid graph $H$ with cell size $\ell = 1$, and perform the puzzle piece modification. This combined solver is called $\mathrm{TSP}_{\mathrm{Cov}}$. As shown in Section 4, we can limit the worst-case overhead for performing turns of $\mathrm{TSP}_{\mathrm{Cov}}$ to $\tau = 0.32566$ per length of the tour, or about 32.6 %.

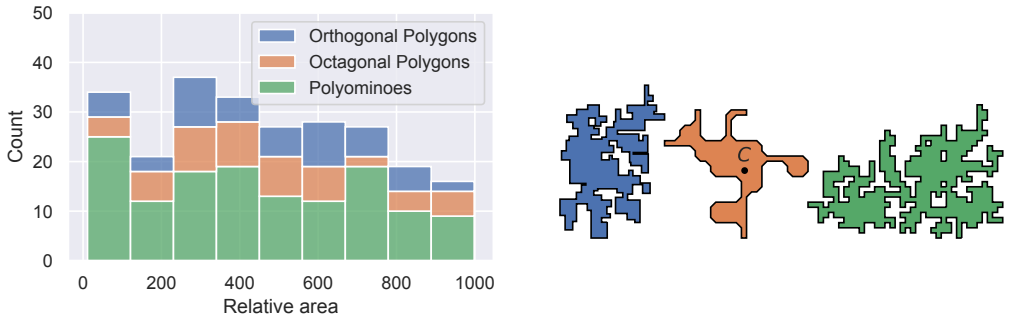
### 5.1.4 Improving the Tour

For a feasible tour from $\mathrm{TSP}_{\mathrm{Cov}}$, we use an LNS-algorithm [41], which iteratively fixes a large part of the IP and only optimizes a small region of tiles; this yields $\mathrm{TSP}_{\mathrm{Turn}}$. We select a random tile from the current tour and a fixed number of adjacent pixels. This yields a limited-size integer program, in which only the involved puzzle pieces are allowed to change. To escape local minima, we tune the size (and runtime) of the IP after each iteration based on the runtime of the previous iteration. In the end, we attempt to solve the IP on the complete instance, using the start solution from the LNS. This provides lower bounds on the best placement of puzzle pieces.

## 5.2 Experimental Setup

Our practical implementation was tested on a workstation with an AMD Ryzen 7 5800X ($8 \times 3.8$ GHz) CPU and 128 GB of RAM. The code and data are publicly available[1]. We used the *srpg_iso*, *srpg_iso_aligned* and *srpg_octa* instances and generated additional polyominoes with the open-source code from the Salzburg Database of Geometric Inputs [19].

[1] https://github.com/tubs-alg/lawn-mowing-from-algebra-to-algorithms




**(a)** Distribution of instances. **(b)** Three instances with the cutter $C$.

**Figure 4** Examples of the used polygons and their size distribution.

See Figure 4 for the overall distribution and Figure 12 in the full version [27] for examples. We considered polygons with up to $n = 300$ vertices and a cutter with diameter 1. Overall, this resulted in 327 instances. All experiments were carried out with a maximum runtime of 300 s for TSP, LNS and final IP computation. To solve the TSP efficiently, we used the python binding *pyconcorde* of the Concorde TSP Solver [43]. All components of $\text{TSP}_{\text{Cov}}$ and $\text{TSP}_{\text{Turn}}$ were implemented in Python 3.10 and used the IP solver Gurobi (v10.0) [30]. As in previous work [26] the *relative area* (ratio of convex hull area of $P$ and cutter area $A(C)$) is more significant for the difficulty of an instance than number of vertices of $P$.

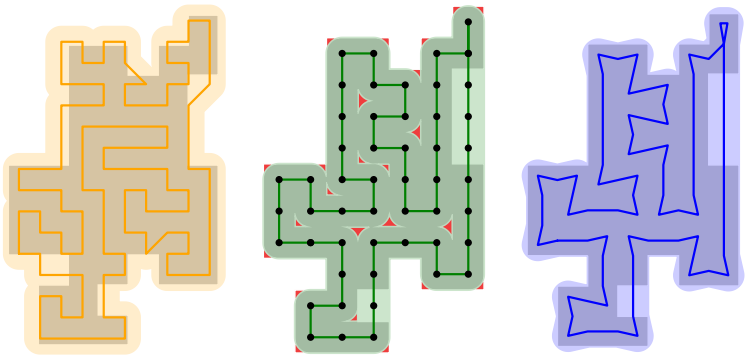


**Figure 5** (Left) A $\text{TSP}_{\text{Small}}$ tour yields a feasible but expensive LMP tour. (Middle) A TSP tour of the underlying dual grid graph, with uncovered patches shown in red. (Right) A feasible LMP tour after puzzle piece modification of the TSP tour.

## 5.3 Evaluation

We discuss our practical results along a number of research questions (RQ).

**RQ1: How does $\text{TSP}_{\text{Cov}}$ compare to $\text{TSP}_{\text{Small}}$ in practice?** We compared the worst-case bound of 32.6 % for $\text{TSP}_{\text{Cov}}$ to the actual performance, using the total cost of $\text{TSP}_{\text{Small}}$ as a baseline. See Figure 5 for an example and Figure 6 for the average relative modification cost. This shows not more than an additional 19 % cost, with only small variation over size and

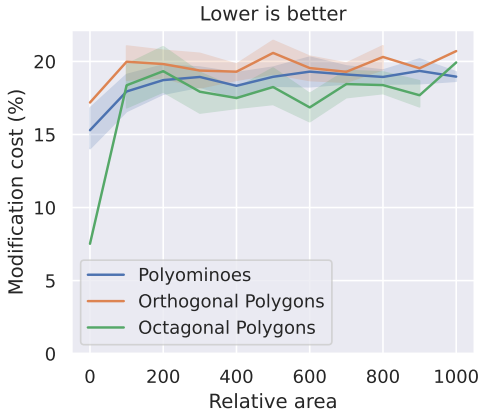



**Figure 6** Modification cost over size and instance type. The modification induces a cost of around 19 % over all instance types and sizes. The plot shows the average modification cost and the 95 % confidence interval.

type. Figure 7b shows that the practical average reduction from $\text{TSP}_{\text{Small}}$ is independent of the size of the polygon, but differs strongly for the different instance classes; we save $\approx 27\,\%$ for polyominoes, $\approx 24\,\%$ for octagonal polygons, and $\approx 5\,\%$ for orthogonal polygons.

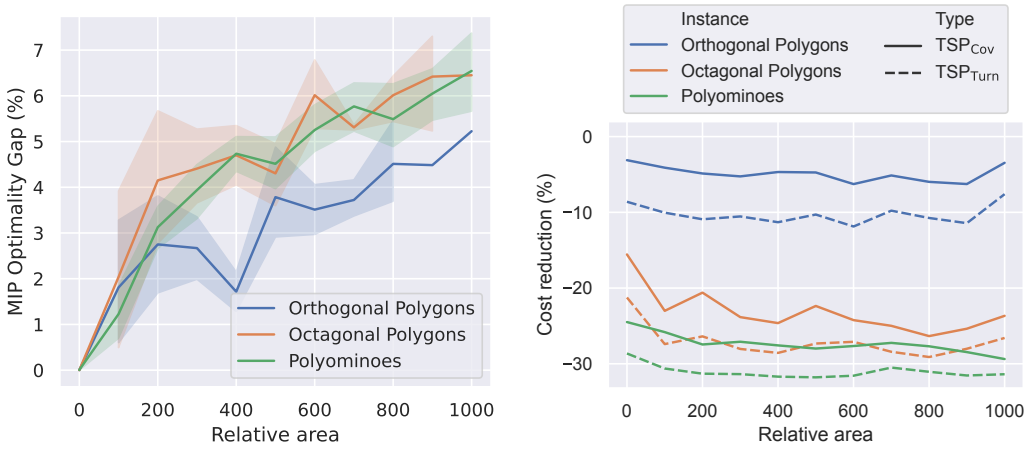
**RQ2: How good are the solutions achieved by $\text{TSP}_{\text{Turn}}$?** For the considered large instances, provably optimal solutions for the turn-cost minimizing IP are hard to find, so we considered the remaining optimality gap in the IP. Figure 7a shows that gaps remain below 7 % even for large instances, and below 5 % on average for medium-sized instances.

We also compared the tours from $\text{TSP}_{\text{Cov}}$ with the cheapest tours obtained by $\text{TSP}_{\text{Turn}}$ and $\text{TSP}_{\text{Small}}$. As shown in Figure 7b, on average we obtain $\approx 5\,\%$ shorter tours when compared to the $\text{TSP}_{\text{Cov}}$ tours, independent of instance size and type. For orthogonal polygons, this doubles the cost reduction.

**RQ3: How far are we from the geometric area lower bound?** A remaining gap between $\text{TSP}_{\text{Turn}}$ and the area bound may result from two sources, both from (i) the quality of the upper bound (and thus $\text{TSP}_{\text{Turn}}$) and (ii) the quality of the area lower bound, for the following reasons. (i) The optimal LMP tour is not restricted to the grid graph $H$, so there may be cheaper tours than what we obtain from $\text{TSP}_{\text{Turn}}$. (ii) The simple area bound (corresponding to Lemma 9) is relatively weak, so it is conceivable that a serious gap to this lower bound remains.

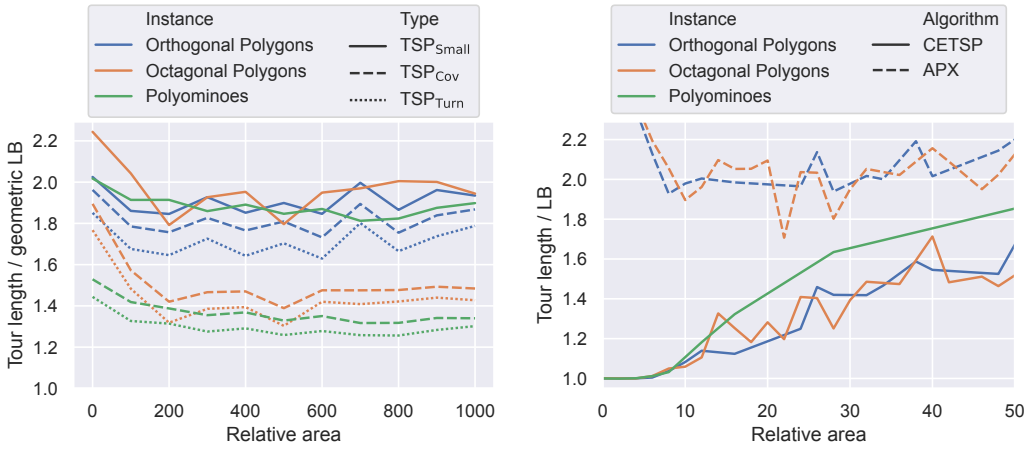
Overall, the combination of both effects remains limited, as can be seen from Figure 8 (showing the ratio of $\text{TSP}_{\text{Turn}}$ value and area bound): For the octagonal polygons and polyominoes, we are on average at most 50 % above the area bound. For orthogonal polygons, the relative gap is on average below 80 %.

**RQ4: How do our solutions compare to previous practical work?** As shown before, our results are already considerably better than work based on $\text{TSP}_{\text{Small}}$. A comparison to the previous best practical results by Fekete et al. [26] (whose instances were used as a subset of our benchmarks) is shown in Figure 8; plotted are the ratios between the achieved solution values and the respective lower bounds. Fekete et al. [26] employ a more sophisticated lower bound based on an evaluation of a series of Close-Enough TSP (CETSP) instances. The




**(a)** Optimality gap of the IP. **(b)** Cost reduction.

**Figure 7** (a) Remaining average optimality gaps for the integer program and the 95 % confidence interval. (b) Comparison of the average cost reduction for different approaches and polygon types.

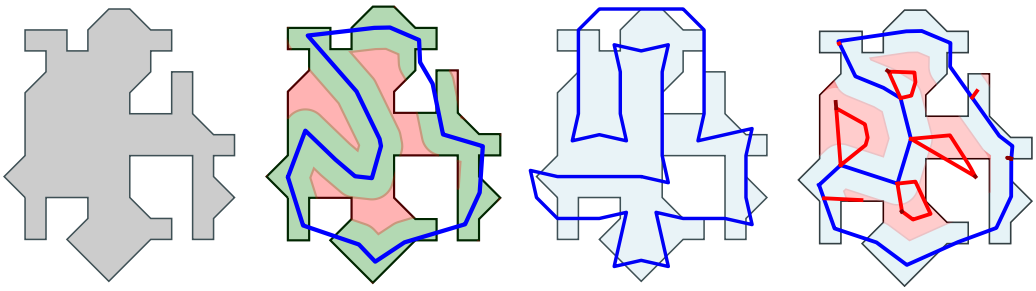



**(a)** Tour length compared to the area bound. **(b)** Solution quality of [26].

**Figure 8** (a) Tour length compared to the (weaker) area lower bound in terms of the average ratio. For octagonal polygons and polyominoes, we can get below 50 % on average. (b) Comparable results for the average solution quality of [26] based on a (stronger) CETSP bound; here APX denotes the performance of the approximation algorithm by [4]. Note the considerably larger relative area in comparison to [26].

authors pointed out that the lower bound computation becomes very expensive even for instances with relative area smaller than 50, see Figures 11 and 12 in [26]. Because we evaluate much larger instances, our ratios only use the relatively straightforward area bound. As a consequence, the denominators of these ratios favor the evaluation for [26], which are shown in Figure 8b; see Figures 9a and 9b for a comparison on a relatively small example that was also shown in [26]. In addition, we were able to achieve results for instances with a relative area 20 times larger than [26].

Despite these additional challenges (of weaker bounds and larger instance sizes), our results compare favorably to the ones reported by [26]. The main reason lies in our structurally simpler approach that still yields good results when the complex evaluation of the CETSP from [26] reaches computational limitations. As can be seen from a comparison of computed trajectories for the visual example (Figures 9c and 9d), this is also reflected in simpler trajectories obtained from $\text{TSP}_{\text{Turn}}$.



**(a)** Area (LB): 36.25. **(b)** LB: 40.94 [26]. **(c)** $\text{TSP}_{\text{Turn}}$ (UB): 66.71. **(d)** UB: 68.16 [26].

**Figure 9** Comparison of $\text{TSP}_{\text{Turn}}$ with lower and upper bounds from Fekete et al. [26].

## 6 Conclusion

We have presented new insights for the Lawn Mowing Problem, starting with an algebraic analysis of the structure of optimal trajectories. As a consequence, we can pinpoint a particular source of the perceived overall difficulty of the problem, and prove that constructing optimal tours necessarily involves operations that go beyond simple geometric means; we can also use these insights to come up with better construction methods for tours, both on the theoretical and the practical side, with minimizing overall turn cost playing a crucial role.

Our results also clear the way for a number of important followup questions. Is it possible to improve our approach for polyominoes? As discussed in the text, considering higher-order connectivity between turns and using slightly off-center, axis-parallel strips appears to be a relatively easy way for (albeit marginal) improvement. It may very well be that this ultimately leads to optimal tours for polyominoes; however, final success on this fundamental challenge will require another breakthrough in establishing lower bounds, as neither the polygon area (which may incur a gap from the optimal value, similar to the number of vertices in a grid graph does from a TSP solution) nor the Close-Enough TSP bound for a finite set of witness points may suffice to certify optimality. Given that an optimal tour may also involve portions that are not axis-parallel, it will also require further algebraic analysis of turns that are not multiples of 90 degrees.

For the Lawn Mowing Problem on general regions (which may not even have to be connected), our hardness result hints at further difficulties. It is quite conceivable that the general LMP is not just algebraically hard, but even $\exists\mathbb{R}$-complete. Even in that case, we believe that further engineering of the tile-based mowing of polyominoes (with attention to turn cost) and Close-Enough TSP may be the most helpful tools for further systematic improvement.

## References


1 David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 2007. doi:10.1016/j.orl.2007.06.002.

2 Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005. doi:10.1137/S0097539703434267.

3 Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. The lawnmower problem. In *Canadian Conference on Computational Geometry (CCCG)*, pages 461–466, 1993. URL: https://cglab.ca/~cccg/proceedings/1993/Paper79.pdf.

4 Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17:25–50, 2000. doi:10.1016/S0925-7721(00)00015-8.

5 Esther M. Arkin, Martin Held, and Christopher L. Smith. Optimization problems related to zigzag pocket machining. *Algorithmica*, 26(2):197–236, 2000. doi:10.1007/s004539910010.

6 Rik Bähnemann, Nicholas Lawrance, Jen Jen Chung, Michael Pantic, Roland Siegwart, and Juan Nieto. Revisiting Boustrophedon coverage path planning as a generalized traveling salesman problem. In *Field and Service Robotics*, pages 277–290, 2021. doi:10.1007/978-981-15-9460-1_20.

7 Chanderjit Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(2):177–191, 1988. doi:10.1007/BF02187906.

8 Michael J. Bannister, William E. Devanny, David Eppstein, and Michael T. Goodrich. The galois complexity of graph drawing: Why numerical solutions are ubiquitous for force-directed, spectral, and circle packing drawings. *Journal of Graph Algorithms and Applications*, 19(2):619–656, 2015. doi:10.7155/jgaa.00349.

9 Aaron T. Becker, Mustapha Debboun, Sándor P. Fekete, Dominik Krupke, and An Nguyen. Zapping zika with a mosquito-managing drone: Computing optimal flight patterns with minimum turn cost. In *Symposium on Computational Geometry (SoCG)*, pages 62:1–62:5, 2017. Video at https://www.youtube.com/watch?v=SFyOMDgdNao. doi:10.4230/LIPIcs.SoCG.2017.62.

10 Károly Bezdek. *Körök optimális fedései (Optimal Covering of Circles)*. PhD thesis, Eötvös Lorand University, 1979.

11 Károly Bezdek. Über einige optimale Konfigurationen von Kreisen. *Ann. Univ. Sci. Budapest Rolando Eötvös Sect. Math*, 27:143–151, 1984.

12 Richard Bormann, Joshua Hampp, and Martin Hägele. New brooms sweep clean - an autonomous robotic cleaning assistant for professional office cleaning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4470–4477, 2015. doi:10.1109/ICRA.2015.7139818.

13 Tauã M. Cabreira, Lisane B. Brisolara, and Paulo R. Ferreira Jr. Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1):4, 2019. doi:10.3390/drones3010004.

14 Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, Megan Owen, and Michiel H. M. Smid. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, 2014. doi:10.1016/j.comgeo.2014.02.004.

15 Howie Choset. Coverage for robotics–a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1):113–126, 2001. doi:10.1023/A:1016639210559.

16 Howie Choset and Philippe Pignon. Coverage path planning: The Boustrophedon cellular decomposition. In *Field and Service Robotics*, pages 203–209, 1998. doi:10.1007/978-1-4471-1273-0_32.

17 International Business Machines Corporation. IBM ILOG CPLEX Optimization Studio, 2023.

18 George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. doi:10.1287/opre.2.4.393.

19 Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. Salzburg database of polygonal data: Polygons and their generators. *Data in Brief*, 31:105984, 2020. doi:10.1016/j.dib.2020.105984.

20 Gershon Elber and Myung-Soo Kim. Offsets, sweeps and Minkowski sums. *Computer-Aided Design*, 31(3), 1999. doi:10.1016/S0010-4485(99)00012-3.

21 Brendan Englot and Franz Hover. Sampling-based coverage path planning for inspection of complex structures. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 29–37, 2012. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4728.

22 Gábor Fejes Tóth. Thinnest covering of a circle by eight, nine, or ten congruent circles. *Combinatorial and Computational Geometry*, 52:361–376, 2005. URL: http://library.msri.org/books/Book52/files/18fejes.pdf.

23 Sándor P. Fekete, Utkarsh Gupta, Phillip Keldenich, Christian Scheffer, and Sahil Shah. Worst-case optimal covering of rectangles by disks. In *Symposium on Computational Geometry (SoCG)*, pages 42:1–42:23, 2020. doi:10.4230/LIPIcs.SoCG.2020.42.

24 Sándor P. Fekete, Phillip Keldenich, and Christian Scheffer. Covering rectangles by disks: The video. In *Symposium on Computational Geometry (SoCG)*, pages 71:1–75:5, 2020. doi:10.4230/LIPIcs.SoCG.2020.75.

25 Sándor P. Fekete and Dominik Krupke. Practical methods for computing large covering tours and cycle covers with turn cost. In *Algorithm Engineering and Experiments (ALENEX)*, pages 186–198, 2019. doi:10.1137/1.9781611975499.15.

26 Sándor P. Fekete, Dominik Krupke, Michael Perk, Christian Rieck, and Christian Scheffer. A closer cut: Computing near-optimal lawn mowing tours. In *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 1–14, 2023. doi:10.1137/1.9781611977561.ch1.

27 Sándor P. Fekete, Dominik Krupke, Michael Perk, Christian Rieck, and Christian Scheffer. The lawn mowing problem: From algebra to algorithms, 2023. doi:10.48550/arXiv.2307.01092.

28 Erich Friedman. Circles covering squares web page. https://erich-friedman.github.io/packing/circovsqu, 2014. Online, accessed January 10, 2023.

29 Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013. doi:10.1016/j.robot.2013.09.004.

30 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

31 Martin Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *LNCS*. Springer, 1991. doi:10.1007/3-540-54103-9.

32 Martin Held, Gábor Lukács, and László Andor. Pocket machining based on contour-parallel tool paths generated by means of proximity maps. *Computer-Aided Design*, 26(3):189–203, 1994. doi:10.1016/0010-4485(94)90042-6.

33 Aladár Heppes and Hans Melissen. Covering a rectangle with equal circles. *Periodica Mathematica Hungarica*, 34(1-2):65–81, 1997. doi:10.1023/A:1004224507766.

34 Israel N. Herstein. *Topics in algebra*. John Wiley & Sons, 1991.

35 Katharin R. Jensen-Nau, Tucker Hermans, and Kam K. Leang. Near-optimal area-coverage path planning of energy-constrained aerial robots with application in autonomous environmental monitoring. *IEEE Transactions on Automation Science and Engineering*, 18(3):1453–1468, 2021. doi:10.1109/TASE.2020.3016276.

36 Johannes B. M. Melissen and Peter C. Schuur. Covering a rectangle with six and seven circles. *Discrete Applied Mathematics*, 99(1-3):149–156, 2000. doi:10.1016/S0166-218X(99)00130-4.

37 Ghulam Murtaza, Salil S. Kanhere, and Sanjay K. Jha. Priority-based coverage path planning for aerial wireless sensor networks. In *IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (IPSN)*, pages 219–224, 2013. doi:10.1109/ISSNIP.2013.6529792.

38 Eric H. Neville. On the solution of numerical functional equations. *Proceedings of the London Mathematical Society*, 2(1):308–326, 1915. doi:10.1112/plms/s2_14.1.308.

**39** David Nistér, Richard I. Hartley, and Henrik Stewénius. Using galois theory to prove structure from motion algorithms are optimal. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. doi:10.1109/CVPR.2007.383089.

**40** Timo Oksanen and Arto Visala. Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*, 26(8):651–668, 2009. doi:10.1002/rob.20300.

**41** David Pisinger and Stefan Ropke. Large neighborhood search. *Handbook of metaheuristics*, pages 99–127, 2019. doi:10.1007/978-3-319-91086-4_4.

**42** Gokarna Sharma, Ayan Dutta, and Jong-Hoon Kim. Optimal online coverage path planning with energy constraints. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 1189–1197, 2019. URL: https://dl.acm.org/doi/10.5555/3306127.3331820.

**43** Solver, Concorde TSP. Concorde TSP Solver, 2023. URL: https://www.math.uwaterloo.ca/tsp/concorde.html.

**44** Xiaoming Zheng, Sven Koenig, David Kempe, and Sonal Jain. Multirobot forest coverage for weighted and unweighted terrain. *IEEE Transactions on Robotics*, 26(6):1018–1031, 2010. doi:10.1109/TRO.2010.2072271.

# Learned Monotone Minimal Perfect Hashing

**Paolo Ferragina**
University of Pisa, Italy

**Hans-Peter Lehmann**
Karlsruhe Institute of Technology, Germany

**Peter Sanders**
Karlsruhe Institute of Technology, Germany

**Giorgio Vinciguerra**
University of Pisa, Italy

## Abstract

A Monotone Minimal Perfect Hash Function (MMPHF) constructed on a set $S$ of keys is a function that maps each key in $S$ to its rank. On keys not in $S$, the function returns an arbitrary value. Applications range from databases, search engines, data encryption, to pattern-matching algorithms.

In this paper, we describe LeMonHash, a new technique for constructing MMPHFs for integers. The core idea of LeMonHash is surprisingly simple and effective: we learn a monotone mapping from keys to their rank via an error-bounded piecewise linear model (the PGM-index), and then we solve the collisions that might arise among keys mapping to the same rank estimate by associating small integers with them in a retrieval data structure (BuRR). On synthetic random datasets, LeMonHash needs 34% less space than the next larger competitor, while achieving about 16 times faster queries. On real-world datasets, the space usage is very close to or much better than the best competitors, while achieving up to 19 times faster queries than the next *larger* competitor. As far as the construction of LeMonHash is concerned, we get an improvement by a factor of up to 2, compared to the competitor with the next best space usage.

We also investigate the case of keys being variable-length strings, introducing the so-called LeMonHash-VL: it needs space within 13% of the best competitors while achieving up to 3 times faster queries than the next larger competitor.

**2012 ACM Subject Classification** Theory of computation → Data compression; Information systems → Point lookups

**Keywords and phrases** compressed data structure, monotone minimal perfect hashing, retrieval

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.46

**Related Version** *Extended Version*: `https://arxiv.org/abs/2304.11012` [19]

**Supplementary Material** *Software (Source Code)*: `https://github.com/ByteHamster/LeMonHash` archived at `swh:1:dir:af11424a42cdc2b2c4d8c62d041314c37075e8d9`

**Funding** This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500). PF and GV have been supported by the European Union – Horizon 2020 Program under the scheme "INFRAIA-01-2018-2019 – Integrating Activities for Advanced Communities", Grant Agreement n. 871042, "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics" (`http://www.sobigdata.eu`), by the NextGenerationEU – National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) – Project: "SoBigData.it – Strengthening the Italian RI for Social Mining and Big Data Analytics" – Prot. IR0000013 – Avviso n. 3264 del 28/12/2021, by the spoke "FutureHPC & BigData" of the ICSC – Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing funded by European Union – NextGenerationEU – PNRR, by the Italian Ministry of University and Research "Progetti di Rilevante Interesse Nazionale" project: "Multicriteria data structures and algorithms" (grant n. 2017WR7SHH).

**Acknowledgements** We thank Stefan Walzer for early discussions leading to this paper.

## 1 Introduction

Given a set $S$ of $n$ keys drawn from a universe $[u] = \{0, \ldots, u-1\}$, a *Monotone Minimal Perfect Hash Function* (MMPHF) is a hash function that maps keys from $S$ to their rank, and returns an arbitrary value for keys not in $S$. As the name suggests, such a function is both *perfect* because it has no collisions on $S$, and *minimal* because its output range is $[n]$. Differently from a *Minimal Perfect Hash Function* (MPHF) [4, 12, 16, 26, 40, 42, 44, 50], which maps keys from $S$ bijectively to $[n]$ in any order, and from an *Order-Preserving MPHF* (OPMPHF) [25], which retains a given (arbitrary) order on the keys, an MMPHF takes advantage of the natural order of the universe to rank the keys in $S$ in small space, i.e. without encoding them. Indeed, encoding $S$ needs $\log \binom{u}{n}/n = \Omega(\log \frac{u}{n})$ bits per key, and encoding the ranks via an OPMPHF needs $\log(n!)/n = \Omega(\log n)$ bits per key, whilst an MMPHF may use as few as $\mathcal{O}(\log \log \log u)$ bits per key [2], which was recently proven to be optimal [1]. Throughout this paper, $\log x$ stands for $\log_2 x$, and we use the $w$-bit word RAM model.

MMPHFs have numerous applications [1]. They enable efficient queries both in encrypted data [11] and databases [39, 41]. Further applications can be found in information retrieval, where MMPHFs can be used to index the lexicon [54] or to compute term frequencies [6, 46], and in pattern matching [5, 27, 32], where MMPHFs are applied mostly to integer sequences representing the occurrences of certain characters in a text.

Despite the widespread use of MMPHFs and recent advancements on their asymptotic bounds [1], the practical implementations have not made significant progress in terms of new designs and improved space-time performance since their introduction more than a decade ago [3], with only some exceptions targeting query time [33]. As a matter of fact, the solutions in [3] are very sophisticated and well-optimised, and they offer a vast number of efficient space-time trade-offs that were hard to beat.

In this paper, we offer a fresh new perspective on MMPHFs that departs from existing approaches, which are mostly based on a trie-like data structure on the keys. We build upon recent advances in (learning-based) indexing data structures, namely the PGM-index [20, 24], and in retrieval data structures (or static functions), namely BuRR [14]. The former learns a piecewise linear approximation mapping keys in $S$ to their rank estimate. The latter allows associating a small fixed-width integer to each key in $S$, without storing $S$. We combine these two seemingly unrelated data structures in a surprisingly simple and effective way. First, we use the PGM to monotonically map keys to buckets according to their rank estimate, and we store the global rank of each bucket's first key in a compressed data structure. Second, since the rank estimate of some keys might coincide, we solve such bucket collisions by storing the local ranks of these keys using BuRR. We call our proposal *LeMonHash*, because it *learns* and *leverages* the smoothness of the input data to build a space-time efficient *monotone* MPHF. On the theoretical side, this achieves $\mathcal{O}(1)$ bits per key for inputs which are sufficiently random within buckets – breaking the superlinear lower bound. Practically, on various integer datasets tried, it needs about one-third less space than previous approaches and is an order of magnitude faster. We also extend LeMonHash to support variable-length string keys. This approach needs space within 13% of the best competitors while being up to 3× faster.

**Outline.** We first describe the basic building blocks of LeMonHash in Section 2 and discuss related work in Section 3. In Section 4, we describe LeMonHash for integers and then extend it to variable-length strings in Section 5. In Section 6, we discuss variants and refinements, before proving the space-time guarantees of LeMonHash in Section 7. In Section 8, we present our experiments. In Section 9, we summarise the paper and give an outlook for future work.

## 2 Preliminaries

In this section, we describe the basic building blocks of LeMonHash.

**Bit Vectors.** Given a bit vector of size $n$ and $b \in \{0,1\}$, the $rank_b(x)$ operation returns the number of $b$-bits before position $x$, and the $select_b(i)$ operation returns the position of the $i$th $b$-bit. These operations can be executed in constant time using as little as $o(n)$ bits on top of the bit vector [13, 34], and they have very space-time efficient implementations [30, 38, 52].

**Elias-Fano.** Elias-Fano Coding [15, 17] is a way to efficiently store a non-decreasing sequence of $n$ integers over a universe of size $u$. An integer at position $i$ is split into two parts. The $\log n$ upper bits $x$ are stored in a bit vector $H$ as a 1-bit in $H[i+x]$. The remaining lower bits are directly stored in an array $L$. Integers can be accessed in constant time by finding the $i$th 1-bit in $H$ using a $select_1$ data structure and by looking up the lower bits in $L$. Predecessor queries are possible by determining the range of integers that share the same upper bits of the query key using two $select_0$ queries, and then performing a binary search on that range. If there are no duplicates, this binary search takes $\mathcal{O}\big(\min\{\log n, \log \frac{u}{n}\}\big)$ time. The space usage of an Elias-Fano coded sequence is $n\lceil \log \frac{u}{n} \rceil + 2n + o(n)$ bits (see [47, §4.4]). Partitioned Elias-Fano [49] is an extension that uses dynamic programming to partition the input into multiple independent Elias-Fano sequences to minimise the overall space usage.

**PGM-index.** The PGM-index [24] is a space-efficient data structure for predecessor and rank queries on a sorted set of $n$ keys from an integer universe $[u]$. Given a query $q \in [u]$, it computes a rank estimate that is guaranteed to be close to the correct rank by a given integer parameter $\varepsilon$. If one stores the input keys, then the correct rank can be recovered via an $\mathcal{O}(\log \varepsilon)$-time binary search on $2\varepsilon + 1$ keys around the rank estimate. The PGM is constructed in $\mathcal{O}(n)$ time by first mapping the sorted integers $x_1, \ldots, x_n$ in $S$ to points $(x_1, 1), \ldots, (x_n, n)$ in a key-position Cartesian plane, and then learning a piecewise linear $\varepsilon$-approximation of these points, i.e. a sequence of $m$ linear models each approximating the rank of the keys in a certain sub-range of $[u]$ with a maximum absolute error $\varepsilon$. The value $m$, which impacts on the space of the PGM, can range between 1 and $m \leq n/(2\varepsilon)$ [24, Lemma 2] depending on the "approximate linearity" of the points. In practice, it is very low and can be proven to be $m = \mathcal{O}\big(n/\varepsilon^2\big)$ when the gaps between keys are random variables from a proper distribution [20]. The time complexity to compute the rank estimate with a PGM is given by the time to search for the linear model that contains the searched key $q$, which boils down to a predecessor search on $m$ integers from a universe of size $u$. For this, there exist many trade-offs in various models of computations [24, 48].

**Retrieval Data Structures.** A *retrieval data structure* or *static function* on a set $S$ of $n$ keys denotes a function $f : S \to \{0,1\}^r$ that returns a specific $r$-bit value for each key. Applying the function on a key not in $S$ returns an arbitrary value. Retrieval data structures take $(1+\eta)rn$ bits, where $\eta \geq 0$ is the *space overhead* over the space lower bound of $rn$ bits.

*MWHC* [43] is a retrieval data structure based on hypergraph peeling, has an overhead $\eta = 0.23$ and can be evaluated in constant time. 2-step MWHC [3] can have a smaller overhead than MWHC by using two MWHC functions of different widths.

The more recently proposed *Bumped Ribbon Retrieval* (BuRR) data structure [14] basically consists of a matrix. The output value for a key can be obtained by multiplying the hash of the key with that matrix. The matrix can be calculated by solving a linear equation

system. Because BuRR uses hash functions with *spacial coupling* [53], the equation system is almost a diagonal matrix, which makes it very efficient to solve. When some rows of the equation system would prevent successful solving, BuRR *bumps* these rows (and the corresponding keys) to the next layer of the same data structure. BuRR has an overhead $\eta = \mathcal{O}\big(\log W/(rW^2)\big)$ and can be evaluated in $\mathcal{O}(1 + rW/\log n)$ time, where $W = \mathcal{O}(\log n)$ is a parameter called ribbon width. In practice, BuRR achieves space overheads well below $\eta = 1\%$ while being faster than widely used data structures with much larger overhead [14].

## 3 Related Work

Non-monotone perfect hash functions are a related and very active area of research [4, 7, 12, 16, 26, 40, 42, 44, 50]. Due to space constraints, we do not review them in detail. For a more detailed list, refer to Ref. [40]. We also do not describe order-preserving minimal perfect hash functions [25] because their theoretical lower bound can trivially be reached by using a retrieval data structure taking $\log n$ bits per key (plus a small overhead). Another loosely related result is using learned models as a replacement for hash functions in traditional hash tables [37, 51], but it generally has a negative impact on the probe/insert throughput (and most likely on the space too, due to the storage of the models' parameters, which these studies do not evaluate). We now look at monotone minimal perfect hash functions, first describing the idea of bucketing before then continuing with specific MMPHF constructions.

**Bucketing.** Bucketing [3] is a general technique to break down MMPHF construction into smaller sub-problems. The idea is to store a simple monotone, but not necessarily minimal or perfect *distributor* function that maps input keys to buckets. Each bucket receives a smaller number of keys that can then be handled using some (smaller) MMPHF data structure. To determine the global rank of a key, we need the prefix sum of the bucket sizes. For equally-sized buckets, this is trivial. Otherwise, this sequence can be stored with Elias-Fano coding. In the paper by Belazzougui et al. [3], where many of the following techniques are described, the authors use MWHC [43] to explicitly store the ranks within each bucket. LeMonHash uses a learned distributor and buckets of expected size 1 (see Section 4).

**Longest Common Prefix.** Bucketing with Longest Common Prefixes (LCP) [2] maps keys to equally sized buckets. A first retrieval data structure maps all keys to the *length* of the LCP among all keys in its bucket. A second one then maps the *value* of the LCP to the bucket index. Overall, it uses $\mathcal{O}(\log \log u)$ bits per key and query time $\mathcal{O}((\log u)/w)$, and in practice it has been shown to be the fastest but the most space-inefficient MMPHF [3].

**Partial Compacted Trie.** First map the keys to equally sized buckets and consider the last key of each bucket as a *router* indexed by a *compacted trie*, e.g., a binary tree where every node contains a bit string denoting the common prefix of its descending keys. During queries, the trie is traversed by comparing the bit string of the traversed nodes with the key to decide whether to stop the search operation at some node (if the prefix does not match), or descend into the left or right subtree based on the next bit of the key. A *Partial Compacted Trie* (PaCo Trie) [3] compresses the compacted trie above by 30–50% by exploiting the fact that, in an MMPHF, the trie needs to correctly rank only the keys from the input set. Therefore, each node can store a shorter bit string just long enough to correctly route all input keys.

**Hollow Trie.** A *Hollow Trie* [3] only stores the *position* of the next bit to look at. Hollow tries can be represented succinctly using balanced parentheses [45]. To use hollow tries for bucketing, and thus allow the routing of not-indexed keys, we need a modification to the data structure. The *Hollow Trie Distributor* [3] uses a retrieval data structure that maps the compacted substrings of each key in each tree node to the behaviour of that key in the node (stopping at the left or right of the node, or following the trie using the next bit of the key). Overall, it uses $\mathcal{O}(\log\log\log u)$ bits per key and query time $\mathcal{O}(\log u)$.

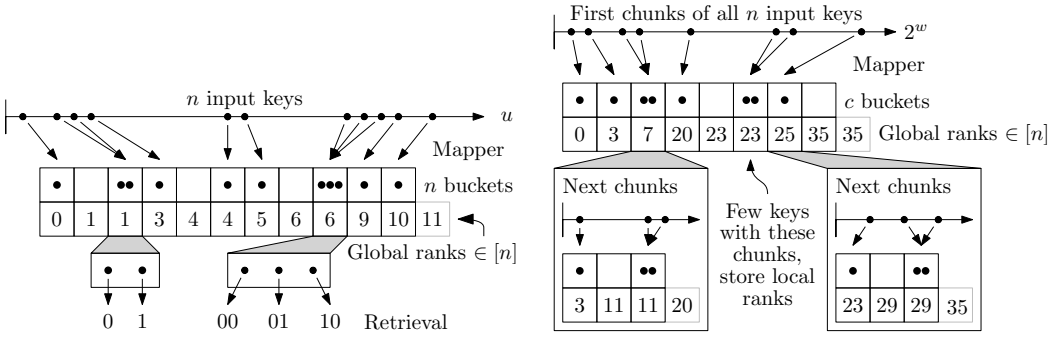
**ZFast Trie.** To construct a *ZFast Trie* [2], we first generate a path-compacted trie. Then, for prefixes of a specific length (*2-fattest number*) of all input keys, a dictionary stores the trie node that represents that prefix. A query can then perform a binary search over the length of the queried key. If there is no node in the dictionary for a given prefix, the search can continue with the pivot as its upper bound. If there is a node, the lower bound of the search can be set to the length of the longest common prefix of all keys represented by that node. The ZFast trie uses $\mathcal{O}(\log\log\log u)$ bits per key and query time $\mathcal{O}((\log u)/w + \log\log u)$.

**Path Decomposed Trie.** In the previous paragraphs, we described binary tries with a rather high height. However, those tries are inefficient to query because of the pointer chasing to non-local memory areas. The main idea behind *Path Decomposed Tries* [18], which can be used as an MMPHF [33], is to reduce the height of the tries. We first select one path all the way from the root node to a leaf. This path is now contracted to a single node, which becomes the root node in our new path decomposed trie. The remaining nodes in the original trie form subtries branching from every node in that path. We take all of these subtries, make them children of the root node, and annotate them by their branching character with respect to the selected path. The subtries are then converted to path decomposed tries recursively. In *centroid* path decomposition, the path to be contracted is always the one that descends to the node with the most leaves in its subtree.

## 4 LeMonHash

We now introduce the main contribution of this paper – the MMPHF LeMonHash. The core idea of LeMonHash is surprisingly simple. We take all the $n$ input integers and map them to $n$ buckets using some *monotone* mapping function, that we will describe later. We store an Elias-Fano coded sequence with the *global ranks* of the first key in each bucket using $2n + o(n)$ bits. Given a bucket of size $b$, we use a $\lceil \log b \rceil$-bit retrieval data structure (see Section 2) to store the *local ranks* of all its keys. Note that we do not need to store local ranks if the bucket has only 0 or 1 keys. For squeezing space, instead of storing one retrieval data structure per bucket, we store a collection of retrieval data structures so that the $i$th one stores the local ranks of all keys mapped to buckets whose size $b$ is such that $i = \lceil \log b \rceil$. An illustration of the overall data structure is given in Figure 1a.

**Bucket Mapping Function.** The space efficiency of LeMonHash is directly related to the quality of the monotone mapping function. For uniform random integers, a linear mapping from input keys to $n$ buckets, i.e. a mapping from a key $x$ to the bucket number $\lfloor xn/u \rfloor$, leads to an MMPHF with a space usage of just 2.915 bits per key (see Theorem 1). Intuitively, such a linear mapping returns a rank estimate in $[n]$ for a given key. However, for skewed distributions, the rank estimate can be far away which can create large buckets whose local ranks are expensive to store. For example, if the majority of the keys are such that $x < u/n$,




**(a)** LeMonHash. Keys are mapped to buckets. Ranks within buckets are stored in (a collection of) retrieval data structures.

**(b)** LeMonHash-VL. Global ranks in each level are stored together. Buckets that are not handled recursively use retrieval data structures like before.

**Figure 1** Illustration of the LeMonHash and LeMonHash-VL data structures.

then the first bucket will be large enough to require $\Theta(\log n)$ bits per key, i.e. our MMPHF degenerates to a trivial OPMPHF. To tackle this problem, we implement the mapping function with a *PGM-index* [24]. As we observed in Section 2, the PGM was originally designed as a predecessor-search data structure. Here, we use the PGM as a *rank estimator* that, for a given key, returns an $\varepsilon$-bounded estimate of its rank. To achieve this result in LeMonHash, we do not store the list of indexed keys and simply use the PGM's rank estimate as the bucket index. The PGM internally adapts to the input data by learning the smoothness in the distribution via a piecewise linear $\varepsilon$-approximation model, thus it can be thought of as a "local" approximation of the linear mapping above. Real-world data sets can often be approximated using piecewise linear models, as discussed in the literature [20] and also demonstrated by the good space efficiency of our experiments (see Section 8). There is a trade-off between the amount of space needed to represent the PGM and the quality of the mapping, which depends on both the input data distribution and the given integer parameter $\varepsilon$. In Section 8, we test both a version with a constant $\varepsilon$ value and a version that auto-tunes its value by constructing multiple PGMs and then selecting the optimal $\varepsilon$. Finally, we observe that with the PGM mapper, unlike for the linear mapping and other non error-bounded learning-based approaches [23, 36], the number of retrieval data structures we need to keep is bounded by $\mathcal{O}(\log \varepsilon)$ regardless of the input key distribution (see Theorem 2).

**Queries.** Given a key $q$, we obtain its bucket $i$ using the mapping function. The global rank of the (first key in the) bucket is the $i$th integer in the Elias-Fano coded sequence of global ranks, which can be accessed in constant time, and the bucket size is computed by subtraction from the next integer in that sequence. The bucket size $b$ directly tells us which retrieval data structure to query, i.e. the $\lceil \log b \rceil$th one. Evaluating the retrieval data structure with $q$ gives us its local rank in the bucket. Adding this to the global rank of the bucket gives us the rank of $q$. As we show in Section 7, for uniform data, the linear bucket mapper gives constant time queries, while for other inputs we use the PGM mapper and the query time is $\mathcal{O}(\log \log u)$.

**Comparison to Known Solutions.** Known MMPHFs in the literature typically divide the keys into equal-size buckets and build a compact trie-based distributor. Unlike them, LeMonHash learns the data linearities and leverages them to distribute keys to buckets close to their rank. Whenever some keys collide into a bucket, LeMonHash handles these keys via

a (small) collection of succinct retrieval structures. In contrast to known solutions, whenever a key is the only one mapped to its bucket, no information needs to be stored in (and no query is issued on) a retrieval data structure. These features allow LeMonHash to possibly achieve reduced space occupancy compared to classic MMPHFs, which are oblivious to data linearities. Also, LeMonHash can reduce the query time by replacing the cache-inefficient traversal of a trie with the PGM mapper, which in practice is fast to evaluate.

## 5 LeMonHash-VL

Of course, the idea of LeMonHash can be immediately applied to keys whose maximum longest common prefix (LCP) is less than $w$ bits. In this case, each string prefix and the following bit (which are sufficient to distinguish every string from each other) fit into one machine word and thus can be handled efficiently in time and in space by the PGM mapper. For strings with longer LCPs, we introduce a tree data structure that we call *LeMonHash-VL* (since it handles Variable-Length strings). The main idea is to simply compute the bucket mapping on a length-$w$ substring of each string, which we call a *chunk*. Buckets that receive many keys using this procedure are then handled recursively. Details follow.

**Overview.** We start with a root node representing all the string keys in $S$ and consider the set of chunks extracted from each key starting from position $|p|$ (which we store), where $p$ is the LCP among the keys in $S$. Given these $c$ distinct chunks, we construct a PGM mapper to distribute the keys to buckets in $[c]$, and we store an Elias-Fano coded sequence with the global ranks of the first key in each bucket. Clearly, different keys can be mapped to the same bucket because the PGM mapper is not perfect (as in the integer case) and because they share the same chunk value (unlike in the integer case). For example, for the strings $S = \{\texttt{cherry}, \texttt{cocoa}, \texttt{coconut}\}$ with $p = \texttt{c}$ and chunks composed of 3 characters, the keys `cocoa` and `coconut` share the chunk value `oco` and will be mapped to the same bucket.

If a bucket of size $b$ contains fewer input strings than a specific threshold $t$, we store the local ranks of the strings in the bucket in a $\lceil \log b \rceil$-bit retrieval data structure. Once again, we do not need to store local ranks if the bucket has only 0 or 1 keys. If instead the bucket is large (i.e. $b \geq t$), we create a *child node* in the tree data structure by applying the same idea recursively on the strings $S'$ of that bucket. This means that we compute a PGM mapper on the chunks extracted from each string in $S'$ starting from position $|p'|$, where $p'$ is the LCP among the bucket strings $S'$. Notice that $|p'| \geq |p|$ but we always guarantee that $S' \subsetneq S$, so the recursion is bounded. In practice, we set the threshold $t = 128$ (see Section 8.1).

At query time, we can use the sequence of global ranks to calculate the bucket size $b$, which allows determining whether we need to continue recursively on a child (because $b \geq t$) or directly return the global rank of the bucket plus the local rank stored in the $\lceil \log b \rceil$-bit retrieval data structure. Figure 1b gives an overview of the data structure.

We observe that the global ranks of each node increase monotonically from left to right in each level of the overall tree. Therefore, we merge all these global ranks in a level into one Elias-Fano sequence, thereby avoiding the space overhead of storing many small sequences.

Of course, each inner node of the tree needs some extra metadata, like the encoding of its bucket mapper, the value of $|p|$, and an offset to its first global rank in the per-level Elias-Fano sequence. We associate a node to its metadata via a minimal perfect hash function, where the identifier of a node is given by the path of the buckets' indices leading to it.

Given the overall idea, there is a wide range of optimisations that we use. In the following, we outline the main algorithmic ones and refer the interested reader to our implementation [28] and the extended version [19] for the many other small-and-tricky optimisations, such as the use of specialised instructions like `popcount` and `bextr`, or lookup tables.

**Alphabet Reduction.** The number of nodes and the depth of LeMonHash-VL depend on both the length and distribution of the input strings, and on how well the PGM mapper at each node can map strings to distinct buckets given their $w$-bit chunks. Therefore, we should aim to fit as much information as possible in the $w$-bit chunks. We do so by exploiting the fact that, in real-world data sets, often only a very small alphabet $\Sigma$ of branching characters distinguish the strings in each bucket, and that we do not care about the other characters. We extract chunks from the suffix of each string starting from the position following the LCP $p$, as before, but interpret the suffix as a number in radix $\sigma = |\Sigma|$ where each character is replaced by its 0-based index in $\Sigma$ if present, or by 0 if not present. For example, for a node on the strings $\{\texttt{shoppers}, \texttt{shopping}, \texttt{shops}\}$ whose LCP is $p = \texttt{shop}$, we would store the alphabet $\Sigma = \{\texttt{e}, \texttt{i}, \texttt{p}, \texttt{s}\}$ and map the suffix "`pers`" of "`shoppers`" to $\text{index}(\texttt{p})\sigma^3 + \text{index}(\texttt{e})\sigma^2 + \text{index}(\texttt{r})\sigma^1 + \text{index}(\texttt{s})\sigma^0 = 2\sigma^3 + 0\sigma^2 + 0\sigma^1 + 3\sigma^0$. Observe that the chunks computed in this way still preserve the lexicographic order of the strings. The number of characters we extract is computed to fit as many characters as possible in a $w$-bit word, i.e. $\lfloor w / \log \sigma \rfloor$ characters. In our implementation over bytes, we store $\Sigma$ via a bitmap of size 128 or 256, depending on whether its characters are a subset of ASCII or not. Finally, we mention that a mapping from strings to numbers in radix $\sigma$ has also been used to build compressed string dictionaries [8], but the twist here is that we are considering only the alphabet of the branching characters since we do not need to store the keys.

**Elias-Fano Sequences.** The large per-level Elias-Fano sequences of global ranks have a very irregular structure. For example, if many of the strings in a node share the same chunks, there is a large gap between two of the stored ranks. We can deal with these irregularities and reduce the overall space usage by using partitioned Elias-Fano [49]. Furthermore, the PGM mappers do not always provide a very uniform mapping, which thus results in empty buckets. An empty bucket corresponds to a duplicate offset value being stored in the Elias-Fano sequences (see e.g. the duplicate offset 23 in Figure 1b). To optimise the space usage of such duplicates, we filter them out before constructing the partitioned Elias-Fano sequence. We do this by grouping the stored numbers in groups of 3 numbers. If all 3 numbers are duplicates of the number before that group, we do not need to store the group. A bit vector with rank support indicates which groups were removed.

**Perfect Chunk Mapping.** In many datasets, there might be only a small number of different chunks, even if the number of strings they represent is large. For instance, chunks computed on the first bytes of a set of URLs might be a few due to the scarcity of hostnames, but each host may contain many distinct pages. In these cases, instead of a PGM, it might be more space-efficient to build a (perfect) map from chunks to buckets in $[c]$ via a retrieval data structure taking $c\lceil \log c \rceil$ bits overall (plus a small overhead), where $c$ is the number of distinct chunks. In practice, we apply this optimisation whenever $c < 128$ (see Section 8.1).

**Comparison to Known Solutions.** In essence, LeMonHash-VL applies the idea of LeMonHash recursively to handle variable-length strings. Therefore, unlike known solutions, it can leverage data linearities to distribute $w$-bit chunks from the input strings to buckets using small space, and use additional child nodes only whenever a bucket contains many strings that thus require inspecting the following chunks to be distinguished. Additionally, it performs an adaptive alphabet reduction within the buckets to fit more information in the $w$-bit chunks, thus leveraging the presence of more regularities in the input data. Overall, these features result in a data structure that has a small height and is efficient to be traversed.

## 6 Variants and Refinements

LeMonHash can be refined in numerous ways, which we only mention briefly due to space constraints. Looking at a possible external memory implementation, LeMonHash can be constructed trivially by a linear sweep and queries are possible using a suitable representation of the predecessor and bucket-size data structures. LeMonHash can also be constructed in parallel without affecting the queries, in contrast to the trivial parallelisation by partitioning the input. In LeMonHash-VL, extracting chunks from non-contiguous bytes reduces the height of the trees but has worse trade-offs in practice. Finally, we present an alternative to storing the local ranks explicitly. The idea is to recursively split the universe size of that bucket and record the number of keys smaller than that midpoint. Despite its query overhead, this technique might be of general interest for MMPHFs. Refer to the extended version [19] for details.

## 7 Analysis

We now prove some properties of our LeMonHash data structure for integers. In our analysis, we use succinct retrieval data structures taking $rn + o(n)$ bits per stored value and answering queries in constant time (see Section 2 and [14]). Furthermore, since our bucket mappers need multiplications and divisions, we make the simplifying assumption $u = 2^w$ to avoid dealing with the increased complexity of these arithmetic operations over large integers.

▶ **Theorem 1.** *A LeMonHash data structure with a bucket mapper that simply performs a linear interpolation of the universe on a list of $n$ uniform random keys needs $\approx n(2.91536 + o(1))$ bits on average*[1] *and answers queries in constant time.*

**Proof.** We approximate the number of keys per bucket using a Poisson distribution which results in $0.91536n + o(n)$ bits of space for the retrieval data structures. On top of that, an Elias-Fano coding of the global bucket ranks gives $2n + o(n)$ bits. Refer to the extended version [19] for the full proof. ◀

While this result is formally only valid for a global uniform distribution, for use in LeMonHash it suffices if each segment computed by the PGM-index is sufficiently smooth. It need not even be uniformly random as long as each local bucket has a constant expected size. As long as the space for encoding the segments is in $\mathcal{O}(n)$ bits, we retain the linear space bound of Theorem 1. Moreover, the following worst-case analysis gives us a fallback position that holds regardless of any assumptions.

▶ **Theorem 2.** *A LeMonHash data structure with the PGM mapper takes $n(\lceil \log(2\varepsilon+1) \rceil + 2 + o(1)) + \mathcal{O}\left(m \log \frac{u}{m}\right)$ bits of space in the worst case and answers queries in $\mathcal{O}\left(\log \log_w \frac{u}{m}\right)$ time, where $m$ is the number of linear models in a PGM with an integer parameter $\varepsilon \geq 0$ constructed on the $n$ input keys.*

[1] Numerically, we find that a better space usage of $\approx 2.902n$ bits can be achieved by mapping the $n$ keys to only $\approx 0.909n$ buckets, but this difference is irrelevant in practice. It is also interesting to note that this is close to the space requirements of most of the practical non-monotone MPHFs [4,7,12,16,26,40,42,44,50]. Using an MMPHF can be useful when indexing an array through an MPHF, because sorting the hash values can be more cache efficient than a large number of random accesses to the array.

**Proof.** The basic idea is that the rank estimate returned by the PGM is guaranteed to be far from the correct rank by $\varepsilon$, which limits the space of the retrieval data structures. The $\mathcal{O}\left(m \log \frac{u}{m}\right)$-term in the space bound is given by a compressed encoding of the linear models in the PGM, and the query time is given by a predecessor search structure on the linear models' keys. Refer to the extended version [19] for the full proof. ◀

The worst-case bounds obtained in Theorem 2 are hard to compare with the ones of classic MMPHF (see Section 3) due to the presence of $m$ (and $\varepsilon$), which depends on (and must be tuned according to) the approximate linearity of the input data, which classic MMPHFs are oblivious to.[2] Refer to Section 2 for bounds on $m$. Our experiments show that we obtain better space or space close to the best classic MMPHFs, while being much faster (we use a weaker but practical predecessor search structure than the one in Theorem 2). Refer to Section 8 for details.

## 8 Experiments

In the following section, we first compare different configurations of LeMonHash and LeMonHash-VL before comparing them with competitors from the literature.

**Experimental Setup.** We perform our experiments on an Intel Xeon E5-2670 v3 with a base clock speed of 2.3 GHz running Ubuntu 20.04 with Linux 5.10.0. We use the GNU C++ compiler version 11.1.0 with optimisation flags `-O3 -march=native`. As a retrieval data structure, we use BuRR [14] with 64-bit ribbon width and 2-bit bumping info. To store the bucket sizes, we use the select data structure by Kurpicz [38] in LeMonHash and Partitioned Elias-Fano [49] in LeMonHash-VL. To map tree paths to the node metadata, we use the MPHF PTHash [50]. For the PGM implementation in LeMonHash, we use the encoding from Theorem 2 and use a predecessor search on the Elias-Fano sequence (Section 2). In LeMonHash-VL, since the number of linear models in a node is typically small, we encode them explicitly as fixed-width triples (*key*, *slope*, *intercept*) and find the predecessor via a binary search on the keys. All our experiments are executed on a single thread. Because the variation is very small, we run each experiment only twice and report the average. We run the Java competitors on OpenJDK 17.0.4 and perform one warm-up run for the just-in-time compiler that is not measured. With this, the Java performance is expected to be close to C++ [3]. Because Java does not have an unsigned 64-bit integer type, we subtract $2^{63}$ from each input key to keep their relative order.

The code and scripts needed to reproduce our experiments are available on GitHub under the General Public License [28, 29].

**Datasets.** Our datasets, as in previous evaluations [3, 33], are a *text* dataset that contains terms appearing in the text of web pages [3] and *urls* crawled from .uk domains in 2007 [10]. Additionally, we also test with *dna* sequences consisting of 32-mers [22]. Regarding real-world integer datasets, *5gram* contains positions of the most frequent letter in the BWT of a text file containing 5-grams found in books indexed by Google [9, 31]. The *fb* dataset contains Facebook user IDs [35] and *osm* contains OpenStreetMap locations [35]. As synthetic integer datasets, we use 64-bit *uniform*, *normal*, and *exponential* distributions. Refer to Table 1 for details.

[2] This happens also in other problems in which data is encoded with linear models [9, 21].

**Table 1** Datasets used for the experiments, together with their length or average (ø) length. Top: real-world string datasets. Middle: real-world integer datasets. Bottom: synthetic integer datasets.

| Dataset | $n$ | Length | Description |
|---|---|---|---|
| text | 35M | ø 11 bytes | Terms appearing in the text of web pages, GOV2 corpus [3] |
| dna | 367M | 32 bytes | 32-mer from a DNA sequence, Pizza&Chili corpus [22] |
| urls | 106M | ø 105 bytes | Web URLs crawled from .uk domains in 2007 [10] |
| 5gram | 145M | 32 bits | Positions of the most frequent letter in the BWT of a text file containing 5-grams found in books indexed by Google [9, 31] |
| fb | 200M | 64 bits | Facebook user IDs [35] |
| osm | 800M | 64 bits | OpenStreetMap locations [35] |
| uniform | 100M | 64 bits | Uniform random |
| normal | 100M | 64 bits | Normal distribution ($\mu = 10^{15}$, $\sigma^2 = 10^{10}$) |
| exponential | 100M | 64 bits | Exponential distribution ($\lambda = 1$, scaled with $10^{15}$) |

## 8.1 Tuning Parameters

In the following section, we compare several configuration parameters of LeMonHash and show how they provide a trade-off between space usage and performance.

**LeMonHash.** Different ways of mapping the keys to buckets have their own advantages and disadvantages. Table 2 gives measurements of the construction and query throughput, as well as the space consumption of different bucket mappers. Our implementation of LeMonHash with a linear bucket mapper achieves a space usage of $2.94n$ bits, which is remarkably close to the theoretical space usage of $2.91n$ bits (see Theorem 1). Of course, a global, linear mapping does not work for all datasets. A bucket mapper that creates equal-width segments by interpolating between sampled keys (denoted as "Segmented" in the table) is fast to construct and query, and it achieves good space usage. But, as for the global linear mapping, this approach is not robust enough to manage arbitrary input distributions. In particular, for this heuristic mapper, it is easy to come up with a worst-case input that degenerates the space usage. Conversely, with the PGM mapper, LeMonHash still achieves $2.96n$ and $2.98n$ bits on uniform random integers but it is more performant and robust on other datasets (except on osm, where the heuristic mapper obtains a good enough mapping with only its equal-width segments, which are inexpensive to store). In fact, we explicitly avoided heuristic design choices in our PGM mapper (such as sampling input keys, removing outliers, or using linear regression) to not inflate our performance on the tested datasets at the expense of robustness on unknown ones (see Ref. [36]). Finally, on most input distributions, auto-tuning the value of $\varepsilon \in \{15, 31, 63\}$ does not have a large effect on the space usage.

**LeMonHash-VL.** Table 3 lists the effect of alphabet reduction on the query and construction performance. In general, alphabet reduction enables noticeable space improvements with only a small impact on the construction time. For the dna dataset, which uses only 15 different characters, the alphabet reduction has the largest effect, saving 1.3 bits per key and simultaneously making the queries 40% faster. The faster queries can be explained by the reduced tree height. Note that alphabet reduction makes the queries slightly slower for the other datasets. The reason is that instead of one single `bswap` instruction for chunk extraction, it needs multiple arithmetic operations (including `popcount`) for each input character. The

**Table 2** Comparison of different bucket mappers. The space usage is given in bits per key, the query throughput in kQueries/second, and the construction throughput (c.t.) in MKeys/second.

| Dataset | Linear mapper | | | PGM $\varepsilon = \text{auto}$ | | | PGM $\varepsilon = 31$ | | | Segmented | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | bpk | kq/s | c.t. | bpk | kq/s | c.t. | bpk | kq/s | c.t. | bpk | kq/s | c.t. |
| 5gram | 5.60 | 1833.5 | 6.2 | 2.62 | 1747.0 | 3.8 | 2.63 | 1779.4 | 8.5 | 2.64 | 2145.9 | 14.5 |
| fb | 34.35 | 0.8 | 5.1 | 4.91 | 1156.1 | 2.8 | 4.91 | 1150.7 | 5.1 | 4.93 | 1441.3 | 7.2 |
| osm | 12.92 | 1525.3 | 5.5 | 4.42 | 999.6 | 2.8 | 4.42 | 998.6 | 5.0 | 4.33 | 1272.9 | 6.8 |
| uniform | 2.94 | 3244.6 | 8.7 | 2.96 | 1903.3 | 3.5 | 2.98 | 1850.5 | 6.5 | 3.03 | 2192.0 | 8.7 |
| normal | 34.27 | 105.3 | 4.8 | 2.95 | 1935.0 | 3.6 | 2.97 | 1858.0 | 6.6 | 3.00 | 1727.7 | 8.7 |
| exponential | 5.42 | 2715.9 | 6.0 | 2.95 | 1876.9 | 3.6 | 2.98 | 1791.5 | 6.6 | 3.01 | 2085.1 | 8.8 |

**Table 3** Comparison of different variants of LeMonHash-VL. The space usage is given in bits per key, the query throughput in kQueries/second, and the construction throughput (c.t.) in MKeys/second. Variants with and without alphabet reduction (AR), a special indexed variant (Idx, see the extended version [19]), and a variant with fixed instead of auto-tuned parameter $\varepsilon$ for the bucket mapper.

| Dataset | $\varepsilon = \text{auto}$, no AR | | | $\varepsilon = \text{auto}$, AR | | | $\varepsilon = 63$, AR | | | Idx, $\varepsilon = \text{auto}$, AR | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | bpk | kq/s | c.t. | bpk | kq/s | c.t. | bpk | kq/s | c.t. | bpk | kq/s | c.t. |
| text | 6.52 | 1062.9 | 1.7 | 6.03 | 1005.8 | 1.6 | 6.08 | 1001.8 | 2.5 | 6.10 | 933.2 | 2.3 |
| dna | 7.66 | 452.8 | 2.0 | 6.32 | 631.3 | 1.7 | 6.25 | 644.8 | 2.7 | 6.27 | 601.1 | 2.4 |
| urls | 7.14 | 282.7 | 2.3 | 6.37 | 298.8 | 1.8 | 6.46 | 295.1 | 2.3 | 6.63 | 298.1 | 1.6 |

indexed variant that builds chunks from the distinguishing bytes instead of a contiguous byte range (see the extended version [19]) is slower to construct but does not show clear space savings, which can be explained by larger per-node metadata. We also experimented with different thresholds for when to stop recursion, as well as the perfect chunk mapping (see Section 5). Given that the space overhead from each bucket mapper is the same for all data sets, it is not surprising that the same threshold (128 keys) works well for all datasets (see the extended version [19]). Finally, making the $\varepsilon$ value of the PGM mapper constant instead of auto-tuned, we naturally get faster construction. As in the integer case, one would expect a fixed $\varepsilon$ value to always produce results that are the same or worse than the auto-tuned version. This is not the case because, in the recursive setting, it is hard to estimate the effect of a mapper on the overall space usage. Therefore, an $\varepsilon$ value that needs more space locally can lead to a mapping that proves useful on a later level of the tree. This is why $\varepsilon = 63$ can achieve better space usage than the auto-tuned version on the dna dataset.

## 8.2 Comparison with Competitors

In this section, we compare the performance of LeMonHash and LeMonHash-VL with competitors from the literature. Competitors include the C++ implementation by Grossi and Ottaviano [33] of the Centroid Hollow Trie, Hollow Trie, and Path Decomposed Trie. Because that implementation only supports string inputs, we convert the integers to a list of fixed-length strings. We point out that the Path Decomposed Trie crashes at an internal assertion when being run on integer datasets. For the Hollow Trie, we encode the skips with either Gamma or Elias-Fano coding, whatever is better on the dataset. We also include the Java

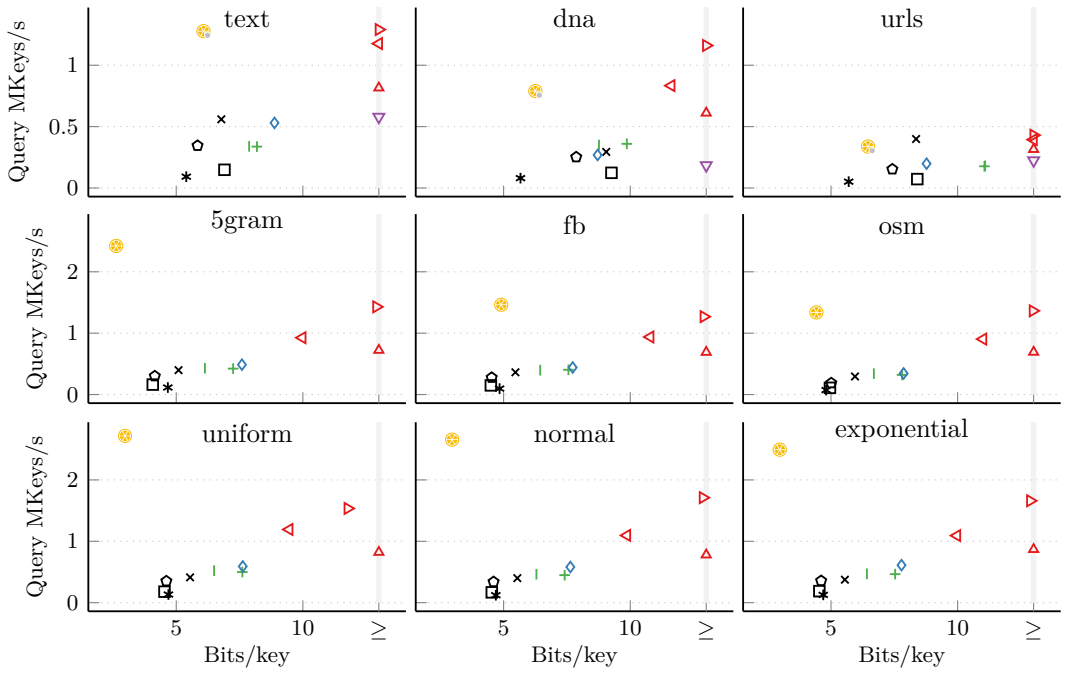



**Figure 2** Query throughput for string, integer, and synthetic integer datasets vs space usage. The top-left corner of every plot shows the top-performing solutions in terms of space-time efficiency.

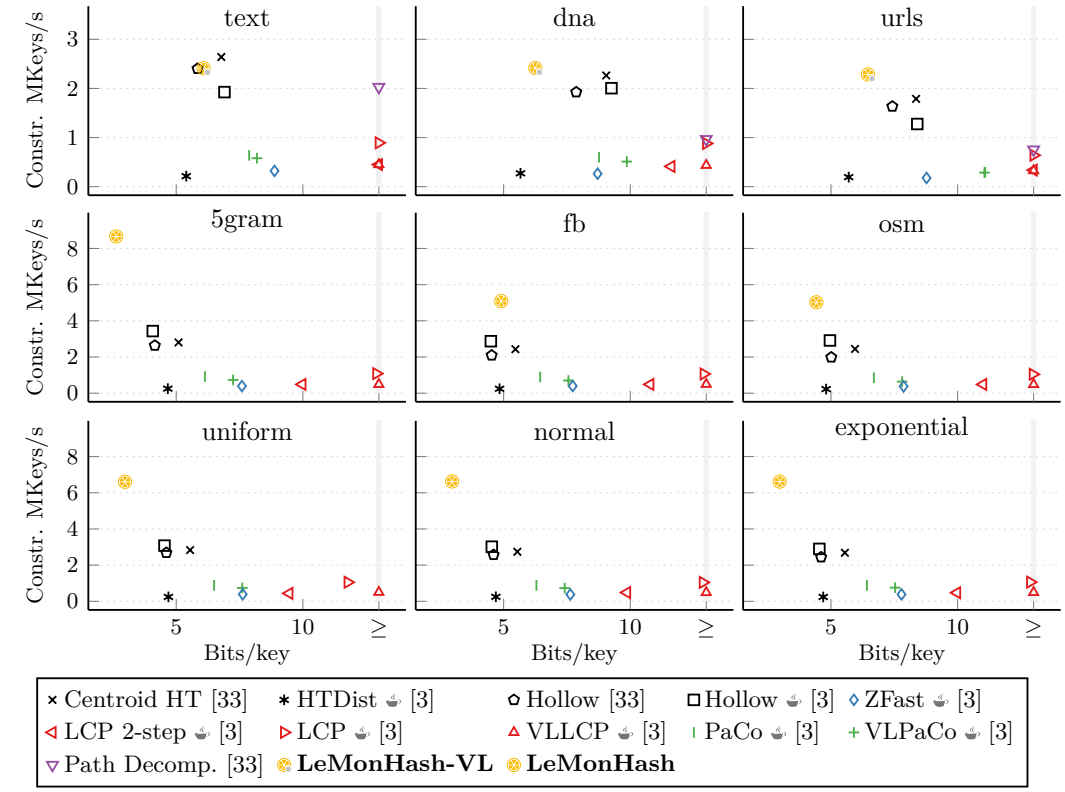



**Figure 3** Construction throughput for string, integer, and synthetic integer datasets. Competitors with the ☕ symbol in the legend are implemented in Java.

implementations by Belazzougui et al. [3] of a range of techniques (see Section 3). We use either the FixedLong or PrefixFreeUtf16 transformation, depending on the data type of the input. For LeMonHash, we use the PGM mapper with $\varepsilon = 31$. For LeMonHash-VL, we use the PGM mapper with $\varepsilon = 63$, alphabet reduction and a recursion threshold $t = 128$.

**Queries.** Figure 2 plots the query throughput against the achieved storage space. In the extended version [19], we additionally detail the numbers in tabular format. The LCP-based methods (see Section 3) have very fast queries but also need the most space (in fact, they appear to the top-right of the plots). At the same time, LeMonHash matches or even outperforms the query throughput of LCP-based methods, while being significantly more space efficient (in fact, it appears towards the top-left of the plots). Compared to competitors with similar space usage, LeMonHash offers significantly higher query throughput.

**Construction.** Figure 3 plots the construction throughput against the space needed. On most synthetic integer datasets, LeMonHash provides a significant improvement to the state-of-the-art approaches, whereas it matches or outperforms the competitors on real-world datasets. LeMonHash improves the construction throughput by up to a factor of 2, compared to the competitor with the next best space usage (typically, variants of the Hollow Trie). While LeMonHash-VL does not achieve the same space usage as the Hollow Trie Distributor, its construction is significantly faster, and still it is the second best in space usage.

## 9 Conclusion and Future Work

In this paper, we have introduced the monotone minimal perfect hash function LeMonHash. LeMonHash, unlike previous solutions, learns and leverages data smoothness to obtain a small space usage and significantly faster queries. On most synthetic and real-world datasets, LeMonHash dominates all competitors – simultaneously – on space usage, construction and query throughput. Our extension to variable-length strings, LeMonHash-VL, consists of trees that are significantly more flat and efficient to traverse than competitors. This enables extremely fast queries with space consumption similar to competitors.

**Future Work.** Many MMPHF construction algorithms are based on the idea of explicitly storing ranks of keys within a small bucket. The idea to split small buckets recursively that we mention in Section 6 can help to reduce the space usage. It remains an open problem whether the idea works in practice, especially when the distribution of keys inside the bucket is skewed. It is also worth investigating a different construction of the piecewise linear approximation in the PGM that minimises the overall space given by the segments *and* the local ranks stored in retrieval data structures, rather than the current approach that maximises the length of the segment (thus minimising just the segments space). Applying non-linear transformations like low-degree polynomials within each segment would also be interesting future work. Finally, it would be interesting to apply smoothed analysis to formally show that many real-world distributions locally behave as if they were uniform random, therefore leading to tighter space bounds.

## References

1 Sepehr Assadi, Martin Farach-Colton, and William Kuszmaul. Tight bounds for monotone minimal perfect hashing. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 456–476, 2023. doi:10.1137/1.9781611977554.CH20.

2 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Monotone minimal perfect hashing: searching a sorted table with $O(1)$ accesses. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–794, 2009. doi:10.1137/1.9781611973068.86.

3 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Theory and practice of monotone minimal perfect hashing. *ACM J. Exp. Algorithmics*, 16, 2011. doi:10.1145/1963190.2025378.

4 Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *Proc. 17th Annual European Symposium on Algorithms (ESA)*, pages 682–693, 2009. doi:10.1007/978-3-642-04128-0_61.

5 Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Trans. Algorithms*, 16(2):17:1–17:54, 2020. doi:10.1145/3381417.

6 Djamal Belazzougui, Gonzalo Navarro, and Daniel Valenzuela. Improved compressed indexes for full-text document retrieval. *J. Discrete Algorithms*, 18:3–13, 2013. doi:10.1016/j.jda.2012.07.005.

7 Dominik Bez, Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. High Performance Construction of RecSplit Based Minimal Perfect Hash Functions. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *LIPIcs*, pages 19:1–19:16, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2023.19.

8 Antonio Boffa, Paolo Ferragina, Francesco Tosoni, and Giorgio Vinciguerra. Compressed string dictionaries via data-aware subtrie compaction. In *Proc. 29th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 233–249, 2022. doi:10.1007/978-3-031-20643-6_17.

9 Antonio Boffa, Paolo Ferragina, and Giorgio Vinciguerra. A learned approach to design compressed rank/select data structures. *ACM Trans. Algorithms*, 18(3):24:1–24:28, 2022. doi:10.1145/3524060.

10 Paolo Boldi, Massimo Santini, and Sebastiano Vigna. A large time-aware web graph. *SIGIR Forum*, 42(2):33–38, 2008. doi:10.1145/1480506.1480511.

11 Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Proc. 31st Annual International Cryptology Conference (CRYPTO)*, pages 578–595, 2011. doi:10.1007/978-3-642-22792-9_33.

12 Jarrod A. Chapman, Isaac Ho, Sirisha Sunkara, Shujun Luo, Gary P. Schroth, and Daniel S. Rokhsar. Meraculous: De novo genome assembly with short paired-end reads. *PLOS ONE*, 6(8):1–13, August 2011. doi:10.1371/journal.pone.0023501.

13 David Richard Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, Canada, 1996.

14 Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. In *Proc. 20th International Symposium on Experimental Algorithms (SEA)*, pages 4:1–4:20, 2022. doi:10.4230/LIPICS.SEA.2022.4.

15 Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974. doi:10.1145/321812.321820.

16 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal perfect hashing via recursive splitting. In *Proc. 22nd Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 175–185, 2020. doi:10.1137/1.9781611976007.14.

17 Robert Mario Fano. On the number of bits required to implement an associative memory. Technical report, MIT, Computer Structures Group, 1971. Project MAC, Memorandum 61".

18 Paolo Ferragina, Roberto Grossi, Ankur Gupta, Rahul Shah, and Jeffrey Scott Vitter. On searching compressed string collections cache-obliviously. In *Proc. 27th ACM Symposium on Principles of Database Systems (PODS)*, pages 181–190, 2008. doi:10.1145/1376916.1376943.

19 Paolo Ferragina, Hans-Peter Lehmann, Peter Sanders, and Giorgio Vinciguerra. Learned monotone minimal perfect hashing, 2023. doi:10.48550/arXiv.2304.11012.

20 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. On the performance of learned data structures. *Theor. Comput. Sci.*, 871:107–120, 2021. doi:10.1016/J.TCS.2021.04.015.

21 Paolo Ferragina, Giovanni Manzini, and Giorgio Vinciguerra. Compressing and querying integer dictionaries under linearities and repetitions. *IEEE Access*, 10:118831–118848, 2022. doi:10.1109/ACCESS.2022.3221520.

22 Paolo Ferragina and Gonzalo Navarro. Pizza&Chili corpus. Accessed: February 2023. URL: http://pizzachili.dcc.uchile.cl/texts.html.

23 Paolo Ferragina and Giorgio Vinciguerra. Learned data structures. In Luca Oneto, Nicolò Navarin, Alessandro Sperduti, and Davide Anguita, editors, *Recent Trends in Learning From Data*, pages 5–41. Springer International Publishing, 2020. doi:10.1007/978-3-030-43883-8_2.

24 Paolo Ferragina and Giorgio Vinciguerra. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *PVLDB*, 13(8):1162–1175, 2020. doi:10.14778/3389133.3389135.

25 Edward A. Fox, Qi Fan Chen, Amjad M. Daoud, and Lenwood S. Heath. Order-preserving minimal perfect hash functions and information retrieval. *ACM Trans. Inf. Syst.*, 9(3):281–308, 1991. doi:10.1145/125187.125200.

26 Edward A. Fox, Qi Fan Chen, and Lenwood S. Heath. A faster algorithm for constructing minimal perfect hash functions. In *Proc. 15th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 266–273, 1992. doi:10.1145/133160.133209.

27 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020. doi:10.1145/3375890.

28 LeMonHash - GitHub. https://github.com/ByteHamster/LeMonHash, 2023.

29 MMPHF-Experiments - GitHub. https://github.com/ByteHamster/MMPHF-Experiments, 2023.

30 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *Proc. 13th International Symposium on Experimental Algorithms (SEA)*, pages 326–337, 2014. doi:10.1007/978-3-319-07959-2_28.

31 Google. Google ngram exports. Accessed: March 2023. URL: https://storage.googleapis.com/books/ngrams/books/datasetsv3.html.

32 Roberto Grossi, Alessio Orlandi, and Rajeev Raman. Optimal trade-offs for succinct string indexes. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 678–689, 2010. doi:10.1007/978-3-642-14165-2_57.

33 Roberto Grossi and Giuseppe Ottaviano. Fast compressed tries through path decompositions. *ACM J. Exp. Algorithmics*, 19(1), 2014. doi:10.1145/2656332.

34 Guy Jacobson. Space-efficient static trees and graphs. In *Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989. doi:10.1109/SFCS.1989.63533.

35 Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. SOSD: A benchmark for learned indexes. *CoRR*, abs/1911.13014, 2019.

36 Evgenios M. Kornaropoulos, Silei Ren, and Roberto Tamassia. The price of tailoring the index to your data: Poisoning attacks on learned index structures. In *Proc. 48th International Conference on Management of Data (SIGMOD)*, pages 1331–1344, 2022. doi:10.1145/3514221.3517867.

37 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proc. 44th International Conference on Management of Data (SIGMOD)*, pages 489–504, 2018. doi:10.1145/3183713.3196909.

38 Florian Kurpicz. Engineering compact data structures for rank and select queries on bit vectors. In *Proc. 29th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 257–272, 2022. doi:10.1007/978-3-031-20643-6_19.

39 Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. PaCHash: Packed and compressed hash tables. In *Proc. 25th Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 162–175, 2023. doi:10.1137/1.9781611977561.ch14.

40 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. SicHash - small irregular cuckoo tables for perfect hashing. In *Proc. 25th Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 176–189, 2022. doi:10.1137/1.9781611977561.ch15.

41 Hyeontaek Lim, Bin Fan, David G. Andersen, and Michael Kaminsky. SILT: a memory-efficient, high-performance key-value store. In *Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–13. ACM, 2011. doi:10.1145/2043556.2043558.

42 Antoine Limasset, Guillaume Rizk, Rayan Chikhi, and Pierre Peterlongo. Fast and scalable minimal perfect hashing for massive key sets. In *Proc. 16th International Symposium on Experimental Algorithms (SEA)*, pages 25:1–25:16, 2017. doi:10.4230/LIPICS.SEA.2017.25.

43 Bohdan S. Majewski, Nicholas C. Wormald, George Havas, and Zbigniew J. Czech. A family of perfect hashing methods. *Comput. J.*, 39(6):547–554, 1996. doi:10.1093/COMJNL/39.6.547.

44 Ingo Müller, Peter Sanders, Robert Schulze, and Wei Zhou. Retrieval and perfect hashing using fingerprinting. In *Proc. 13th International Symposium on Experimental Algorithms (SEA)*, pages 138–149, 2014. doi:10.1007/978-3-319-07959-2_12.

45 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

46 Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):1–47, 2014. doi:10.1145/2535933.

47 Gonzalo Navarro. *Compact data structures: a practical approach*. Cambridge University Press, 2016.

48 Gonzalo Navarro and Javiel Rojas-Ledesma. Predecessor search. *ACM Comput. Surv.*, 53(5), 2020. doi:10.1145/3409371.

49 Giuseppe Ottaviano and Rossano Venturini. Partitioned Elias-Fano indexes. In *Proc. 37th International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 273–282, 2014. doi:10.1145/2600428.2609615.

50 Giulio E. Pibiri and Roberto Trani. PTHash: Revisiting FCH minimal perfect hashing. In *Proc. 44th International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1339–1348, 2021. doi:10.1145/3404835.3462849.

51 Ibrahim Sabek, Kapil Vaidya, Dominik Horn, Andreas Kipf, Michael Mitzenmacher, and Tim Kraska. Can learned models replace hash functions? *PVLDB*, 16(3):532–545, 2022. doi:10.14778/3570690.3570702.

52 Sebastiano Vigna. Broadword implementation of rank/select queries. In *Proc. 7th International Workshop on Experimental Algorithms (WEA)*, pages 154–168. Springer, 2008. doi:10.1007/978-3-540-68552-4_12.

53 Stefan Walzer. Peeling close to the orientability threshold - spatial coupling in hashing-based data structures. In *Proc. 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2194–2211, 2021. doi:10.1137/1.9781611976465.131.

54 Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 2nd edition, 1999.

# Correlating Theory and Practice in Finding Clubs and Plexes

**Aleksander Figiel**
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

**Tomohiro Koana**
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

**André Nichterlein**
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

**Niklas Wünsche**
Unaffiliated Researcher, Berlin, Germany

## Abstract

For solving NP-hard problems there is often a huge gap between theoretical guarantees and observed running times on real-world instances. As a first step towards tackling this issue, we propose an approach to quantify the correlation between theoretical and observed running times.

We use two NP-hard problems related to finding large "cliquish" subgraphs in a given graph as demonstration of this measure. More precisely, we focus on finding maximum $s$-clubs and $s$-plexes, i. e., graphs of diameter $s$ and graphs where each vertex is adjacent to all but $s$ vertices. Preprocessing based on Turing kernelization is a standard tool to tackle these problems, especially on sparse graphs. We provide a parameterized analysis for the Turing kernelization and demonstrate their usefulness in practice. Moreover, we demonstrate that our measure indeed captures the correlation between these new theoretical and the observed running times.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Preprocessing, Turing kernelization, Pearson correlation coefficient

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.47

**Related Version** *Full Version*: `https://arxiv.org/abs/2212.07533`

**Supplementary Material** *Software*: `https://git.tu-berlin.de/afigiel/splex-sclub-correl` archived at `swh:1:dir:12573624f6d53cfd4d7e9181b0cf1596585fb93d`

**Funding** *Aleksander Figiel*: Supported by the Deutsche Forschungsgemeinschaft (DFG) project MaMu (NI 369/19).
*Tomohiro Koana*: Supported by the Deutsche Forschungsgemeinschaft (DFG) project DiPa (NI 369/21).

## 1 Introduction

Highly engineered solvers often perform much better than the known theoretical results would suggest. This is especially true when dealing with NP-hard problems. Unless P = NP, no efficient (i. e. polynomial-time) algorithm exists that solves all input instances correctly. However, optimized implementations can often solve instances with millions of vertices, variables, etc. as demonstrated frequently at algorithm engineering conferences [4, 35]. Of course, these implementations are not polynomial-time algorithms for NP-hard problems – the real-world instances are simply not worst-case instances but have algorithmically exploitable structures. On the other hand, there are usually relatively small instances making these solvers struggle. So theoretical running time bounds can often not predict observed running times on the given data set. Obviously, a better correlation between theoretical results and empirical findings would be highly desirable.


A multivariate (i. e. parameterized) analysis of the algorithm allows for a more nuanced picture of running time bounds. In principle, it could provide us with a much better prediction for the running time. However, a comparison to the theoretical parameterized running time is rarely made in practice (although there are notable exceptions [39, 21]). This is probably due to the multitude of issues arising here; let us mention just a few: For example, most theoretical bounds are stated using the $O$-notation that hides constants. Matching these to observed running times (which depend also on the used hardware) is not straightforward. Moreover, optimized solvers often combine several tricks that work well in different cases. The effect of these tricks is often hard to analyze and the observed running times most likely depend on a large (possibly unknown) set of parameters.

In this work, we use the Pearson correlation coefficient to quantify the correlation between theoretical bounds and observed running times. Our approach allows us to compare which theoretical running-time bound fits "better" to the observed running times for a given data set. Hence, it is relatively easy (but tedious) to generate hypotheses for theoretical bounds that fit well to experimental observations. Although proving such bounds is a different story, our approach can be used to transform practical results into impulses for theory.

We exemplify our approach on the $s$-Club and $s$-Plex problems and show how for various solver variants different theoretical explanations can be used. To this end, we follow the approach of Walteros and Buchanan [39] who demonstrated by means of a multivariate analysis why Clique is often efficiently solvable in relatively sparse graphs.

## 1.1 Related work

**Clique on Sparse Graphs.** Clique is one of Karp's 21 NP-complete problems [18]. As such, it is well studied, both in theory and practice; see Wu and Hao [40] for a survey. The currently fastest exact algorithm has running time $O(1.20^n)$ [41], where $n$ is the number of vertices. While 1.20 seems very small, for a graph with 400 vertices the number of steps has more than 30 *digits* which is still infeasibly large. In contrast, a maximum clique can be found in real-world graphs with millions of vertices [8, 14, 31, 38].

It is easy to see that any clique is contained in the neighborhood of each of its vertices. Thus, a very basic approach solving clique on a sparse graph $G = (V, E)$ is the following. Take a vertex $v$ of minimum degree and find the largest clique in $N[v]$ (the closed neighborhood of $v$). Then, remove $v$ and continue in the same fashion. In the end, output the largest found clique. The *degeneracy* $d$ of a graph is the size of the largest neighborhood encountered in the above algorithm. Hence, the above algorithm can be implemented to run in $1.20^d \cdot n^{O(1)}$ time which is on large sparse graphs far better than the $O(1.20^n)$ bound. Many of the graphs considered by Walteros and Buchanan [39] have several hundred thousand vertices and can be solved in less than a minute (often less than a second). Yet, some of these graphs have a degeneracy of well above 400 (again resulting in an infeasibly large number of steps). To rectify this, Walteros and Buchanan [39] provide an algorithm running in $1.28^g n^{O(1)}$ time, where $g := d - k + 1$ is called the *core-gap* and $k$ denotes the number of vertices in a maximum clique (see Section 3 for a more detailed explanation). Clearly $g$ can be much smaller than $d$. In fact, Walteros and Buchanan [39] observe that all their relatively small but hard-to-solve instances have a large core-gap.

**Clubs and Plexes.** An $s$-club is a graph of diameter $s$. An $s$-plex is a graph with $\ell$ vertices where every vertex has degree at least $\ell - s$. While not required by definition, in this work we only consider *connected* $s$-plexes. The task in $s$-Club / $s$-Plex is to find the largest $s$-club / $s$-plex in a given graph. Both $s$-Club and $s$-Plex are NP-hard as they contain Clique as

special case ($s = 1$). Both problems are well-studied in the literature, both from theoretical and practical perspective. For example, $s$-Plex is W[1]-hard with respect to the parameter solution size $k$ for all $s \geq 1$ [19, 23]. In contrast, if $s > 1$, then $s$-Club is fixed-parameter tractable with respect to the solution size $k$ [33, 9]. We refer to Komusiewicz [22] for a further overview on the parameterized complexity of these problems. Several algorithmic approaches (heuristics and exact algorithms) have been proposed and examined to find maximum-cardinality clubs [6, 5, 7, 9, 15, 28, 32, 25, 24] or plexes [3, 12, 11, 36, 29]. Variants of the above-mentioned Turing kernelization approach are employed in engineered solvers for finding clubs [15, 32] and plexes [3, 29].

## 1.2 Our results

We start by transferring the approach of Walteros and Buchanan [39] to $s$-Club and $s$-Plex. To this end, introducing a new graph parameter, we describe and analyze the Turing kernelization for both problems in Section 3. Moreover, we provide simple branching algorithms showing fixed-parameter tractability with respect to a gap parameter. In Section 4, we then analyze the impact of the Turing kernelization in computational experiments for $s \in \{2, 3\}$. To this end, we use ILP-formulations with and without Turing kernelization and basic lower bounds. For $s$-Club significant speedups are observed whereas for $s$-Plex the improvements are not as clear (though still a speedup factor of more than 2.5 is achieved on average). Finally, in Section 5 we use correlations (more precisely the Pearson correlation coefficient) to analyze how well our theoretical findings fit to our practically observed running times. While this measure makes no statement about the efficiency of the algorithms, we can observe that even with the use of black boxes such as ILP-solvers our theoretical findings are reflected in the experimental results, in particular for the $s$-Club problem.

# 2 Preliminaries

For an integer $a \in \mathbb{N}$, we denote by $[a]$ the set $\{1, \ldots, a\}$. For a graph $G = (V, E)$, let $n := |V|$ and $m := |E|$ be the number of vertices and edges, respectively. Let $u, v \in V$ be two vertices of $G$. Let $\mathrm{dist}_G(u, v)$ denote the length of any shortest path between $u$ and $v$. For $x \in \mathbb{N}$, let $N_{x,G}(v)$ be the $x$th neighborhood of $v$, i. e., the set of vertices $u$ with $1 \leq \mathrm{dist}_G(u, v) \leq x$, $N_{x,G}[v] = \{v\} \cup N_{x,G}(v)$, and $\deg_{x,G}(v)$ be the size of its $x$th neighborhood, i. e., $\deg_{x,G}(v) = |N_{x,G}(v)|$. For a set $X \subseteq V$ of vertices, let $G[X]$ denote the subgraph induced by $X$. We drop the subscript $\cdot_x$ for $x = 1$. Also, we omit the subscript $\cdot_G$ when $G$ is clear from context.

**Clique relaxations.** Let $X$ be a set of vertices. If the vertices of $X$ are pairwise adjacent, then we say that $X$ is a *clique*. Let $s \in \mathbb{N}$ be an integer. We say that $X$ is an *$s$-club* if the vertices of $X$ have pairwise distance at most $s$, i. e., $\max_{u,v \in X} \mathrm{dist}_{G[X]}(u, v) \leq s$ and that $X$ is an *$s$-plex* if $G[X]$ is connected and every vertex $v$ in $X$ has at most $s - 1$ vertices nonadjacent to $v$ in $X \setminus \{v\}$, i. e., $\max_{v \in X} |X \setminus N(v)| \leq s$. (Note that a 1-club and a 1-plex are each a clique.) We sometimes abuse these terms to refer to the subgraph induced by an $s$-club or $s$-plex. The decision problems $s$-Club and $s$-Plex ask, given a graph $G$ and an integer $k \in \mathbb{N}$, whether $G$ contains an $s$-club and $s$-plex, respectively, of size at least $k$.

**Degeneracy.** We say that a graph $G = (V, E)$ is *$d$-degenerate* if for every subgraph $G'$ of $G$, there exists a vertex with $\deg_{G'}(v) \leq d$. Equivalently, $G$ is $d$-degenerate if there is an ordering of $V$ in which every vertex has at most $d$ neighbors that appear later in the ordering.

We say that such an ordering is a *degeneracy ordering* of $G$. The degeneracy $d_G$ of $G$ is the smallest number $d$ such that $G$ is $d$-degenerate. For a vertex $v \in V$ and an ordering $\sigma$ of $V$, we denote by $Q_G^\sigma(v)$ (and $Q_G^\sigma[v]$) the set of vertices in $N_G(v)$ (and $N_G[v]$) that appear after $v$ in $\sigma$. We also omit the superscript $\cdot^\sigma$ when it is clear.

**Parameterized complexity.** Here, we list several relevant notions from parameterized complexity. See e. g., Cygan et al. [13] for a more comprehensive exposition of parameterized complexity. A parameterized problem is *fixed-parameter tractable* or *FPT* for short if every instance $(I, k)$ can be solved in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some computable function $f$. Such an algorithm is called an *FPT algorithm*. It is widely believed that a parameterized problem is not FPT if it is *W[i]-hard* for $i \in \mathbb{N}$. One way to show fixed-parameter tractability is via the notion of *Turing kernel*. For $t \in \mathbb{N}$, a *$t$-oracle* for a parameterized problem is an oracle that solves any instance $(I, k)$ in constant time, provided that $|I| + k \leq t$. We say that a parameterized problem admits a Turing kernel of size $f(k)$ if there is an algorithm with access to an $f(k)$-oracle that solves $(I, k)$ in time $(|I| + k)^{\mathcal{O}(1)}$. It is straightforward to turn a Turing kernel into an FPT algorithm by simply replacing an $f(k)$-oracle with a brute-force algorithm. The brute-force algorithm runs in $f'(k)$ time for some computable function $f'$, resulting in an $f'(k) \cdot (|I| + k)^{\mathcal{O}(1)}$-time algorithm.

## 3 Theory

In this section, we provide theoretical analysis of clique relaxations based on the notion of Turing kernels. We first describe in Section 3.1 the algorithm for Clique outlined by Walteros and Buchanan [39], which runs in $1.28^g n^{\mathcal{O}(1)}$ time for the gap $g := d - k + 1$. The algorithms have two components. The first component is the Turing kernel parameterized by the degeneracy $d$. In short, we show that Clique is polynomial-time solvable when we have access to $f(d)$-oracle (see Section 2). In practice, there is no such convenient oracle so we have to provide some algorithm – this is the second component. One way to substitute the oracle is to use a brute-force algorithm. Since every oracle call takes an input whose size is bounded by $d$, we already obtain an FPT algorithm parameterized by $d$. We can actually make a more refined analysis by considering the gap parameter $g = d - k + 1$. Essentially, we use an FPT algorithm parameterized by $g$ rather than relying on brute force. Walteros and Buchanan [39] showed that many Clique instances that can be solved efficiently in practice indeed have small values of $g$.

We want to adapt this approach to clique relaxations, namely, $s$-Club and $s$-Plex. However, there is one issue: Under standard complexity assumptions, there is no FPT algorithm for $s$-Club or $s$-Plex. More precisely, $s$-Club is known to be NP-hard for $s = 2$ and $d = 6$ [16] and $s$-Plex is known to be W[1]-hard when parameterized by $d + s$ [20]. For this reason, we consider a broader notion of degeneracy, which we call *$x$-degeneracy* for $x \in \mathbb{N}$ (1-degeneracy coincides with the standard degeneracy). We give the formal definition in Section 3.2. With the notion of $x$-degeneracy at hand, we describe how to adapt the approach employed by Walteros and Buchanan [39] to $s$-Club and $s$-Plex in Section 3.3.

### 3.1 Algorithm for Clique

We subsequently sketch the algorithmic approach of Walteros and Buchanan [39] for Clique.

**Turing kernel.** The Clique problem admits a Turing kernel, in which every input to the oracle has at most $d+1$ vertices (thus size $\mathcal{O}(d^2)$) as follows: For an instance $(G, k)$ of Clique, consider a degeneracy ordering of $\sigma = (v_1, \ldots, v_n)$ of $G$. We will assume that $k \leq d + 1$ since

a $d$-degenerate graph has no clique of size $d+2$. Observe that for every clique $C$ of $G$, we have $C \subseteq Q^\sigma[v]$, where $v \in C$ is the vertex that appears first in a degeneracy ordering $\sigma$. Thus, $G$ has a clique of size $k$ if and only if there exists a vertex $v \in V$ such that $G[Q^\sigma[v]]$ has a clique of size $k$. Since $G$ is $d$-degenerate, $G[Q^\sigma[v]]$ has at most $d+1$ vertices and size $\mathcal{O}(d^2)$. This leads to a Turing kernel for the parameter $d$: Simply call the oracle for the instances $(Q^\sigma[v_1], k), \ldots, (Q^\sigma[v_n], k)$ and return yes iff at least one oracle answer was yes.

**Oracle algorithm.** Every oracle can be replaced with a brute-force algorithm running in $\mathcal{O}(2^d d^2)$ time: Since $Q^\sigma[v]$ is of size at most $d+1$, there are $\mathcal{O}(2^d)$ subsets of $Q^\sigma[v]$ and for every subset, it takes $\mathcal{O}(d^2)$ time to check if every pair of vertices are adjacent. Thus, Clique can be solved in $\mathcal{O}(2^d \cdot d^2 n)$ time. In fact, we can refine the analysis for the oracle algorithm in terms of the *gap* parameter $d-k+1$: To that end, we solve the Deletion to Clique problem: Given a graph $G$ and an integer $\ell$, the task is to find a set of at most $\ell$ vertices whose deletion results in a clique. There is a simple $\mathcal{O}(2^\ell n^2)$-time algorithm for this problem: If all vertices are pairwise adjacent and $\ell \geq 0$, then we have a yes-instance at hand. Otherwise, there exist two nonadjacent vertices, say $u$ and $v$. If $\ell = 0$, then we can conclude that there is no solution. If $\ell \geq 1$, then we recursively solve two instances $(G-u, \ell-1)$ and $(G-v, \ell-1)$. This algorithm runs in $\mathcal{O}(2^\ell \cdot n^2)$ time. Since we need to solve this problem on $G[Q^\sigma(v)]$ with $\ell := |Q^\sigma[v]| - k \leq d-k+1$, we have an $\mathcal{O}(2^{d-k+1} \cdot d^2 n)$-time algorithm for Clique. We remark that an instance $(G, \ell)$ of Deletion to Clique is equivalent to the Vertex Cover instance $(\overline{G}, \ell)$ where $\overline{G}$ is the complement graph of $G$. Thus, using a faster known FPT algorithm for Vertex Cover [10], we obtain an $\mathcal{O}^*_d(1.28^{d-k+1} n)$-time algorithm for Clique ($\mathcal{O}^*_d$ hides factors polynomial in $d$).

## 3.2 Extending degeneracy

As mentioned in the beginning of this section, we consider a broader notion of degeneracy that can defined in two ways.

▶ **Definition 3.1.** *Let $G$ be a graph and $x \in \mathbb{N}$. The* $x$-degeneracy *of $G$ is the smallest integer $d_x \in \mathbb{N}$ such that for each subgraph $G'$ of $G$, there exists a vertex $v$ with $|N_{x,G'}(v)| \leq d_x$.*

▶ **Definition 3.2.** *Let $G$ be a graph and $x \in \mathbb{N}$. The* $x$-degeneracy *of $G$ is the smallest integer $d_x$ such that there is an ordering $\sigma = (v_1, \ldots, v_n)$ of $G$ such that for every $i \in [n]$, the $x$th neighborhood of $v_i$ in $G[v_i, \ldots, v_n]$ has size at most $d_x$. The ordering $\sigma$ is called an* $x$-degeneracy ordering. *The set of vertices in $N_{x,G}[v]$ that appear after $v$ in $\sigma$ is $Q^\sigma_{x,G}[v]$.*

It is not difficult to show that these two definitions are equivalent. We remark that the notion of 2-degeneracy has been proposed by Trukhanov et al. [36] in the context of finding $s$-plexes. The $x$-degeneracy and an $x$-degeneracy ordering can be found in polynomial time:

▶ **Theorem 3.3.** *Given a graph $G$ and an integer $x \in \mathbb{N}$, we can compute the $x$-degeneracy of $G$ and an $x$-degeneracy ordering of $G$ in $\mathcal{O}(n^2 m)$ time.*

**Proof.** Repeat the following until the graph is empty: for every vertex $v$, compute the $x$th neighborhood of $v$. Find a vertex whose $x$th neighborhood has the smallest size and delete it from the graph. The ordering in which vertices are deleted is an $x$-degeneracy ordering. The $x$-degeneracy is the maximum over all vertices of the $x$th neighborhood size when they are deleted. Note that we spend $\mathcal{O}(nm)$ time to compute the $x$th neighborhood of every vertex using e.g., BFS. Since we repeat this $n$ times, the algorithm runs in the claimed time. ◀

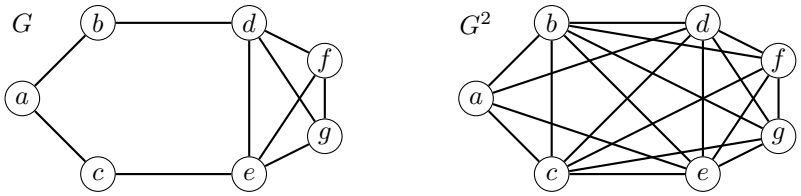



**Figure 1** An example showing the difference between the 2-degeneracy of $G$ (which is 5) and the degeneracy of $G^2$ (which is 6). The ordering $a, b, \ldots, g$ is a 2-degeneracy ordering in $G$ and a degeneracy ordering in $G^2$. Note that $\{b, c, \ldots, g\}$ forms a clique in $G^2$ but not a 2-club in $G$.

We remark that a very similar graph parameter has been studied in the context of $x$-Club [25, 24]: the degeneracy of the $x^{\text{th}}$ power graph $G^x$ of $G$. The power graph $G^x$ has the same vertices as $G$. Moreover, two vertices $u$ and $v$ are adjacent in $G^x$ if and only if $u$ and $v$ have distance at most $x$ in $G$. Note that every $s$-club in $G$ is a clique in $G^x$. The converse, however, is not true; see Figure 1 for an example. While the degeneracy of $G^x$ is in general larger than the $x$-degeneracy, it can be computed faster: in $O(nm)$ time [32]. In fact, the degeneracy of $G^x$ has already been used in Turing kernelization for finding $s$-clubs [32]. Subsequently, we use the smaller $x$-degeneracy for stronger algorithmic results.

## 3.3 Algorithm for $s$-Club and $s$-Plex

**Turing kernel.** For $s$-Club, we adapt the Turing kernel for Clique as follows. For every $s$-club $C$ of $G$, we have $C \subseteq Q_s^\sigma[v]$, where $v \in C$ is the first vertex of $C$ in an $s$-degeneracy ordering $\sigma$. Thus, $G$ has an $s$-club of size $k$ if and only if there exists a vertex $v \in V$ such that $G[Q_s^\sigma[v]]$ has an $s$-club of size $k$. By the definition of $x$-degeneracy, we have $|Q_s^\sigma(v)| \leq d_s$. Thus, we have a Turing kernel in which every oracle call involves at most $d_s + 1$ vertices.

**Oracle algorithm.** Again, we can replace every oracle call with a brute-force algorithm. The input to every oracle call has at most $d_s + 1$ vertices and hence there are $2^{d_s+1}$ subsets. Moreover, for every subset, it takes $\mathcal{O}(d_s^3)$ time to determine whether the vertices have pairwise distance at most $s$, resulting in an algorithm running in $\mathcal{O}(2^{d_s} d_s^3)$ time. As in Section 3.1, we can also refine the algorithm substituting for the oracle using the parameter $d_s - k + 1$. To this end, we solve the Deletion to $s$-Club problem: Given a graph $G$ and an integer $\ell$, the task is to find a set of at most $\ell$ vertices whose deletion results in an $s$-club. There is a simple $\mathcal{O}(2^\ell n^3)$-time algorithm for this problem. If $G$ has diameter at most $s$ and $\ell \geq 0$, then we have a yes-instance at hand. Otherwise there exist two vertices, say $u$ and $v$, with $\text{dist}_G(u, v) > s$. If $\ell = 0$, then we can conclude that there is no solution. If $\ell \geq 1$, then we recursively solve two instances $(G - u, \ell - 1)$ and $(G - v, \ell - 1)$. Since it takes $\mathcal{O}(n^3)$ time to compute all pairwise distances, this algorithm runs in $\mathcal{O}(2^\ell \cdot n^3)$ time. Since we need to solve this problem on $G[Q_s^\sigma[v]]$ with $\ell := |Q_s^\sigma[v]| - k \leq d_s - k + 1$, we obtain:

▶ **Theorem 3.4.** *Given the subgraph $G[Q_s^\sigma[v]]$ for every $v \in V$ for an $s$-degeneracy ordering $\sigma$, $s$-Club can be solved in $\mathcal{O}(2^{d_s - k} \cdot d_s^3 n)$ time.*

**Turing kernel.** For $s$-Plex, we will provide two adaptations. First, note that every $s$-plex is also an $s$-club (recall that we only consider connected $s$-plex). Thus the Turing kernel with the parameterization by $d_s$ follows analogously. For another adaptation, we use the fact that any $s$-plex with at least $2s - 1$ vertices have diameter at most two, as observed by Seidman

and Foster [34]: Suppose that two vertices $u$ and $v$ in an $s$-plex $C$ have distance three in $G[C]$. Then, every vertex in $C$ is nonadjacent to either $u$ or $v$. Since for each of $u$ and $v$, there are at most $s-1$ vertices nonadjacent to it, we have $|C| \leq 2s-2$. This leads to a Turing kernel with respect to the parameter $d_2$ when $k \geq 2s-1$. For every $s$-plex of size at least $2s-1$, we have $C \subseteq Q_2^\sigma[v]$, where $v \in C$ is the first vertex of $C$ in a 2-degeneracy ordering $\sigma$. Thus, we have again a Turing kernel where every oracle call involves at most $d_2+1$ vertices. (Small $s$-plex of size at most $2s-2$ can be found in $O(n^{2s-1})$.)

**Oracle algorithm.** Again, we can replace every oracle call on $h$ vertices with a brute-force algorithm. The input to every oracle call has at most $h+1$ vertices and hence there are $2^{h+1}$ subsets. Moreover, for every subset, it takes $\mathcal{O}(h^2)$ time to determine whether it is an $s$-plex, resulting an algorithm running in $\mathcal{O}(2^h h^2)$ time. As in Section 3.1, we can also refine the algorithm substituting for the oracle using the parameter $h-k+1$. To that end, we solve the Deletion to $s$-Plex problem: Given an $n$-vertex graph $G$ and an integer $\ell$, the task is to find a set of at most $\ell$ vertices whose deletion results in an $s$-plex. There is a simple $\mathcal{O}((s+1)^\ell n^2)$-time algorithm for this problem. If $G$ is an $s$-plex and $\ell \geq 0$, then we have a yes-instance at hand. Otherwise there exist a vertex $v$ and $s$ vertices nonadjacent to $v$. If $\ell = 0$, then we can conclude that there is no solution. If $\ell \geq 1$, then we recursively solve $s+1$ instances $(G-v, \ell-1)$ and $(G-u, \ell-1)$ where $u$ is one of $s$ vertices nonadjacent to $v$. Since it takes $\mathcal{O}(n^2)$ time to check if the graph is an $s$-plex, this algorithm runs in $\mathcal{O}((s+1)^\ell \cdot n^2)$ time. Since we need to solve this problem on graphs with $h$ vertices with $\ell := h-k$, we obtain the following:

▶ **Theorem 3.5.** *Given the subgraph $G[Q_s^\sigma[v]]$ for every $v \in V$ for an $s$-degeneracy ordering $\sigma$, $s$-Plex can be solved in time $\mathcal{O}((s+1)^{d_s-k} \cdot d_s^2 n)$ and $\mathcal{O}(n^{2s-1} + (s+1)^{d_2-k} \cdot d_2^3 n)$.*

We remark that for very small $s$, the first term $n^{2s-1}$ can be ignored in practice, because most instances contain an $s$-plex of size at least $2s-1$.

## 4 Experiments

In this section we present the results of our computational experiments for $s$-Club and $s$-Plex for $s \in \{2,3\}$ on a large dataset of real-world graphs. We did not optimize every aspect of the implementations as our goal is to investigate the effect of Turing kernelization and the extent to which our theoretical findings are reflected in the running time (this is discussed in Section 5). We will see, that the Turing kernelization is quite beneficial for $s$-Club but for $s$-Plex the situation is not as clear.

**Setup & Dataset.** All experiments were performed on a machine running Ubuntu 18.04 LTS, with an Intel Xeon® W-2125 CPU and 256GB of RAM. A maximum running time of 1 hour per instance was set. We used Gurobi 8.1 to solve ILP-formulations, limited to a single thread of execution. The program that was used to build the ILP models was implemented in C++ and compiled with g++ 7.5.

The static graphs from the Network Repository [30] were used for all experiments. Graphs for which at least one solver configuration timed out, ran out of memory, or completed in less than 0.05 seconds were omitted. The last case (less than 0.05 s) is to not deal with the effect of noise in the small running time measurements. The resulting dataset consists of 259 graphs, with 1033 vertices on average. The source code is available at `https://git.tu-berlin.de/afigiel/splex-sclub-correl`.

We remark that $s$-Club and $s$-Plex have been solved for small $s$ on much larger graphs within minutes [15, 12, 11]. The reason we focus on smaller graphs is to have a meaningful multivariate analysis. More precisely, we want to see if the running time grows (as suggested by theory) with growing $x$-degeneracy and gap. Having running times for large graphs with small $x$-degeneracy and gap but not for large graphs with large $x$-degeneracy and gap would give misleading results in our analysis in Section 5.

**Solvers.** We used an ILP solver as oracle for $s$-Club and $s$-Plex in the Turing kernelization. The ILP formulations were taken from the literature: For $s$-Plex we used a straight-froward formulation with $\mathcal{O}(n)$ variables and constraints and $\mathcal{O}(n+m)$ non-zeroes[1] [27].

| | |
|---|---|
| maximize: | $y$ |
| subject to: | $x_v \in \{0,1\}, y \in \{0,\dots,n\}$ |
| | $y = \sum_{v \in V} x_v$ |
| $\forall v \in V$: | $\lvert V\rvert(1-x_v) + \sum_{u \in N(v)} x_u \geq y - s$ |

For 2-Club a simplified formulation by Bourjolly et al. [6] was used. It has $\mathcal{O}(n)$ variables, $\mathcal{O}(n^2)$ constraints, and $\mathcal{O}(n^3)$ non-zeroes.

| | |
|---|---|
| maximize: | $\sum_{v \in V} x_v$ |
| subject to: | $x_v \in \{0,1\}$ |
| $\forall u,v \in V, \mathrm{dist}(u,v) > 2$: | $x_u + x_v \leq 1$ |
| $\forall u,v \in V, \mathrm{dist}(u,v) = 2$: | $x_u + x_v \leq 1 + \sum_{c \in N(u) \cap N(v)} x_c$ |

For 3-Club the neighborhood formulation from Almeida and Carvalho [1, 2] was used, with $\mathcal{O}(n)$ binary variables, $\mathcal{O}(m)$ continuous variables, at most $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ constraints and non-zeroes, respectively[2].

| | |
|---|---|
| maximize: | $\sum_{v \in V} x_v$ |
| subject to: | $x_v \in \{0,1\}$ |
| $\forall u,v \in V, \mathrm{dist}(u,v) > 3$: | $x_u + x_v \leq 1$ |
| $\forall u,v \in V, \mathrm{dist}(u,v) \in \{2,3\}$: | $x_u + x_v \leq 1 + \sum_{c \in N(u) \cap N(v)} x_c + \sum_{e \in E_{uv}} z_e$ |
| $\forall e = \{a,b\} \in E$: | $z_e \leq x_a,\ z_e \leq x_b$ |
| $\forall e \in E$: | $0 \leq z_e \leq 1$ |

[1] An $s$-plex of size $\ell > 2s-1$ is guaranteed to be connected and of diameter two [34]. As we only consider $s \in \{2,3\}$, we do not add constraints enforcing connectedness to the ILP. Unsurprisingly, all found subgraphs were still connected.

[2] For compact and general ILP formulations for $s$-Club ($s \geq 2$) we refer to Salemi and Buchanan [32] and Veremyev et al. [37].

**Table 1** Average running times in seconds of various solver configurations.

| | noTK | full | default | hint |
|---|---|---|---|---|
| 2club | 294.3 | 18.7 | 1.9 | 0.9 |
| 3club | 641.8 | 296.0 | 81.4 | 41.7 |
| 2plex | 26.4 | 63.8 | 10.1 | 10.2 |
| 3plex | 39.6 | 260.6 | 81.4 | 77.6 |
| 3plex-2 | 39.6 | 63.4 | 12.4 | 12.0 |

where $E_{uv} = \{\{p, q\} \in E \mid p \in N(u) \setminus N(v), q \in N(v) \setminus N(u)\}$. We remark that different ILP-formulations for these problems are discussed in the literature [3, 27, 32]. However, analyzing them is beyond the scope of this work.

**Solver variants.** We tested several approaches using these ILP models to cover all concepts discussed in Section 3. To this end, we use four different solver configurations, namely `noTK`, `full`, `default` and `hint` (described below). We will refer to, for example, `2club_noTK` as the benchmark results of the `noTK` solver configuration on the 2-Club problem.

The `noTK` variant (no Turing kernel) simply built a single ILP model for the entire graph. All other variants use the Turing kernelization to some extent. The `full` variant makes only basic use of Turing kernelization, utilizing the 2/3-degeneracy as described in Section 3. There, each oracle call is solved via an ILP. For 2-Club and 2-Plex the Turing kernelization using 2-degeneracy is employed, for 3-Club and 3-Plex the one using 3-degeneracy. As there is only one connected 3-plex of diameter three (the $P_4$) which was never the largest 3-plex in our experiments we also used the 2-degeneracy based Turing kernelization for 3-Plex. We report the results of this variant under `3plex-2` (thus `3plex-2_noTK` is the same as `3plex_noTK`).

The `default` variant uses the Turing kernel approach in combination with a simple lower bound: It uses the maximum solution size of already solved ILPs as a lower bound on the global solution size by adding a constraint to the ILPs enforcing that the solution has to be larger than the current lower bound. Note that this lower bound does not appear in the algorithm descriptions in Section 3. The order in which the ILPs are solved can therefore have an impact on the overall running time. We used a fixed 2/3-degeneracy ordering and did not analyze the impact of the order. Instead, we remove this effect in the `hint` variant. There, we added a constraint to each ILP model in the Turing kernels which enforced that the solution size to the Turing kernel is at least the size of the optimal solution size for the entire graph. Thus, one can think of (heuristically) optimizing in the `default` variant the order in the Turing kernelization so that the oracle calls giving the largest results come first. Alternatively, this shows the maximum speedup possible by a "perfect" lower-bound heuristic. Note that in the `hint` variant at least one ILP model still has a feasible solution.

**Results.** In Table 1 we summarize the average running time of the different solver configurations on the four considered problems. The approach without Turing kernels is significantly slower for 2- and 3-Club but not so much for 2- and 3-Plex. This can also be seen in the detailed comparisons in Figure 2. Interestingly, the `noTK` variants are much faster in finding plexes than in finding clubs (more than 10 times larger average running time). However, the average running time of `2club_default` is five times smaller than that of `2plex_default`. Thus, on the one hand the ILP-formulation we use for finding 2/3-clubs may have a room for

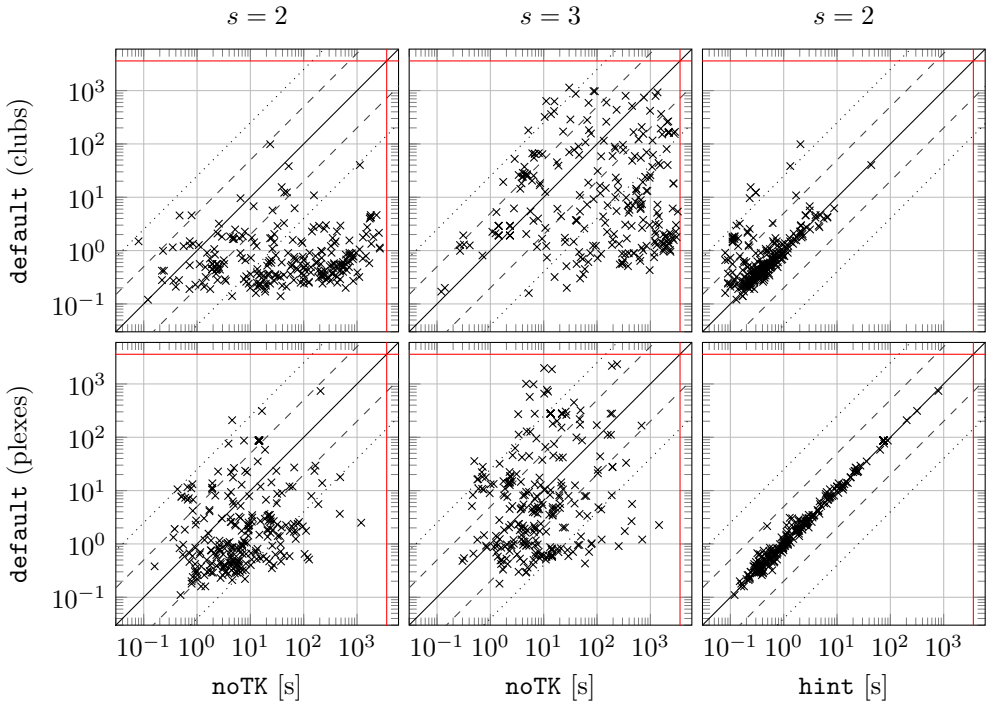



**Figure 2** Running time comparison (in seconds) of different variants (top row for 2- and 3-Club, second row for 2- and 3-Plex). Each cross represents one instance with the $x$- and $y$-coordinates indicating the running time of the respective variant (in seconds): `default` and `noTK` resp. `hint`. Thus, a cross above (below) the solid diagonal indicates that the solver on the $x$-axis ($y$-axis) is faster on the corresponding instance. The diagonal lines mark factors of 1 (solid), 5 (dashed) and 25 (dotted). The solid horizontal red lines indicate the time limit (1 hour). For 2- and 3-Club a significant running time improvement is visible. For 2- and 3-Plex the picture is not so clear.

improvement. On the other hand, the Turing kernel approach works much better for clubs than plexes. The reason is most likely that the Turing kernels are built based on distance, which fits better with clubs than plexes. This can be seen in the gap-parameter: for 73 resp. 57 instances the gap-parameter was zero for 2-Club resp. 3-Club whereas for the plexes the gap-parameter was never zero.

Unsurprisingly, the `hint` variant is the fastest one. However, the `default` variant is not much slower than `hint` (see also right column of Figure 2), even though it does not receive the solution size as input, and instead uses the maximum solution of the previously solved ILPs to update the lower bound. Moreover, the `default` variant is considerably faster than the `full` variant. This shows the strength of providing lower bounds to the ILP-solver. Remarkably, for finding 2/3-clubs even the `full` variant brings a decent speedup compared to the `noTK` variant.

## 5 Correlation between Theory and Practice

We now analyze correlations between the theoretical running time bounds in Section 3 and the measured running times in Section 4. Since we have NP-hard problems, our working hypothesis is that the running time should depend exponentially on some parameter(s).

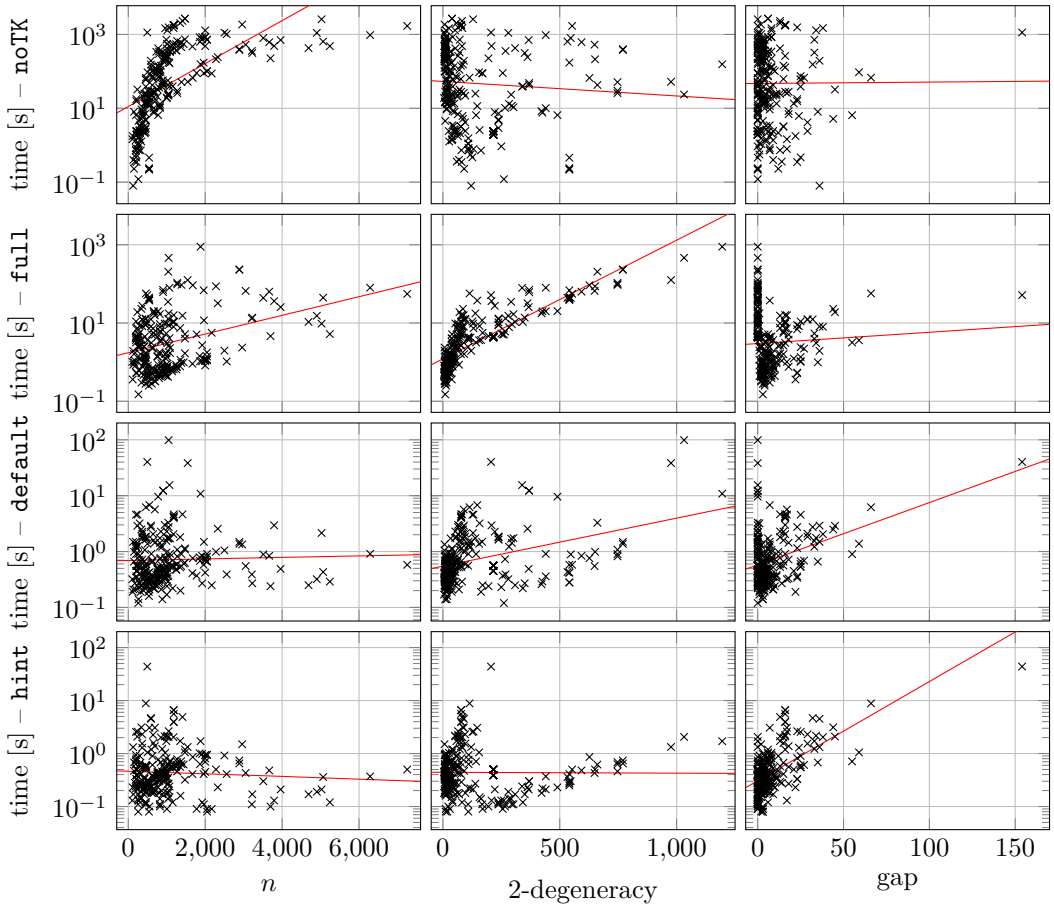



**Figure 3** The running times of `2club_noTK`, `2club_full`, `2club_default`, and `2club_hint` plotted against the number $n$ of vertices, the 2-degeneracy, and the gap of the input graph. The solid red lines are linear regressions best fitting to the data points (where the logarithm of the running times is taken).

Natural parameter candidates are the number $n$ of vertices, the 2- resp. 3-degeneracy, and the gap parameter (suggested by our theoretical findings). Note that there is a multitude of other viable parameters[3]; however, studying them is beyond the scope of this work.

We studied five problems (counting `3plex` and `3plex-2` as two) with four different solvers of each problem. Thus, there are $5 \cdot 4 \cdot 3 = 60$ different (parameter, running time)-pairs to analyze. We illustrate the 12 pairs for `2club` in Figure 3 with the red lines depicting the exponential function of the form $\alpha^p \cdot \beta$ that best fit the data; these lines are computed via linear regression (logarithm of the running time versus parameter value $p$). Obviously, the suggested running time function on the bottom left (`2club_hint` with parameter $n$) is useless: our implementation will in general not become faster the larger the input gets. The red lines on the top left and bottom right seem far more sensible.

[3] See for example `https://manyu.pro/assets/parameter-hierarchy.pdf` or `https://www.graphclasses.org/` for dozens of graph parameters.

## 5.1 Method

Instead of "carefully looking" at each of the plots in Figure 3 and finding arguments for each one of them, we want an automated way of distinguishing useful from useless suggestions. To this end, we suggest using the Pearson correlation coefficient which we subsequently just call correlation coefficient. It is a standard measure of linear correlation between two sets of data. Simply put, given the (running time, parameter) data points the correlation coefficient computes a number between -1 and 1. If there is no correlation at all, then the coefficient is 0. With perfect correlation (i. e. the data points are on a straight line with positive slope) the coefficient is 1. With perfect negative correlation (i. e. the data points are on a straight line with negative slope) it is -1.

We report correlations between the logarithm of the running time and the parameter (in our case either the number of vertices, the generalized degeneracy, or the gap). Thus, values close to 1 represent good correlations (i.e. some exponential dependency between the running time and the parameter). However, note that a better correlation value does not imply a better running time. There are several specifics with our measure that we like to address before reporting the results.

**Exponential Dependency.** Note that our approach "ignores" the base of the exponential functions, in the following sense: A "perfect correlation" of 1 with, say the gap parameter $g$, implies that the measured running time $t$ satisfies the following linear relation for some constant $a$ and $b$: $\log t = a \cdot g + b \iff t = 2^{a \cdot g + b} = \beta \cdot \alpha^g$ where $\beta = 2^b$ and $\alpha = 2^a$ are again two constants. Of course, $\alpha$ has to be nowhere close to the bases of the exponential functions we describe in Section 3. Thus, a "perfect correlation" just indicates that there is *some* exponential dependency between the parameter and the running time.

Our justification here is that the measure needs to be somewhat imprecise: The measured running times can be greatly impacted by the configuration of the ILP-solver [17] or by the experimental setup [26]. Hence, hitting the theorized worst-case running time seems rather unlikely. Since we deal with NP-hard problems we expect (despite all heuristic improvements) some exponential function describing the running time. With our setting we can get suggestions for the base of the exponential function from experimental results.

**Restricted Setting.** We restrict ourselves to correlations between one parameter and the running time. While correlations between multiple parameters and the running time are possible, our theoretical results in Section 3 only suggest exponential dependencies between one parameter (2/3-degeneracy or gap) and the running time and not two parameters. Incorporating the polynomial factors of $n$ in the running times of Section 3 is easily doable. However, in our analysis it changed the coefficients only marginally (by less than 5%, usually much less than 1%), probably due to the relatively small size of the graphs. As we are (for now) only interested in simple exponential dependencies, the Pearson correlation coefficient suffices as we can take the logarithm of all measured running times. Note that there are different correlation coefficients that can also measure non-linear correlations and might be better suited to other settings (e. g. when dealing with polynomial-time solvable problems).

**Simplistic Analysis.** We use a very simplistic statistical analysis. For example, we ignore confidence intervals, $p$-values, or similar issues. The reason being that any "good" correlation between a parameter (or a combination of parameters) and the measured running time is only an *indication* for such a correlation. In particular, if some "new" correlations are discovered with this method, then this only gives *suggestions*. Using our theoretical tools, we

**Table 2** Tables summarizing the correlation of different graph parameters $n$ (left table), $d_2/d_3$ (middle table), and the gap $g$ (right table) with the logarithm of the measured running times of various solver configurations (def abbreviates default). In the middle the correlation with 2-degeneracy is shown for `2club, 2plex` and `3plex-2`, and with 3-degeneracy for `3club` and `3plex`).

| | Correlation with $n$ | | | | Correlation with $d_2/d_3$ | | | | Correlation with gap | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | noTK | full | def | hint | noTK | full | def | hint | noTK | full | def | hint |
| 2club | 0.59 | 0.34 | 0.03 | −0.06 | −0.08 | 0.84 | 0.38 | −0.01 | 0.01 | 0.06 | 0.35 | 0.61 |
| 3club | 0.48 | 0.16 | 0.08 | −0.22 | −0.12 | 0.67 | 0.46 | 0.22 | 0.14 | 0.34 | 0.46 | 0.72 |
| 2plex | 0.53 | 0.07 | 0.08 | 0.07 | 0.21 | 0.63 | 0.46 | 0.46 | 0.22 | 0.60 | 0.44 | 0.45 |
| 3plex | 0.52 | 0.06 | −0.02 | −0.01 | 0.02 | 0.65 | 0.40 | 0.39 | 0.02 | 0.63 | 0.39 | 0.38 |
| 3plex-2 | 0.52 | 0.08 | 0.07 | 0.06 | 0.02 | 0.62 | 0.45 | 0.44 | 0.02 | 0.59 | 0.42 | 0.42 |

can prove (e. g. show a running time bound) or disprove (e. g. NP-hardness for a constant parameter value) such suggestions. This ability of theoretical verification allows us to ignore "safety"-features from statistical analysis.

## 5.2 Results

Table 2 summarizes the 60 correlation coefficients of three graph parameters with the logarithm of the measured running times.

Consider the first row corresponding to 2-Club in the three parts of Table 2. The first columns display the correlation with $n$ which is best for the `noTK` variant. This well reflects our observations for the plots in the left column of Figure 3: The `default` and `hint` variant do not display any reasonable correlation with $n$, only `noTK` does to some extent. Similarly, in the right column of Figure 3 the correlations of the `default` and `hint` variant with the gap-parameter are quite decent, but not for the `noTK` variant. Moreover, in the plot for the `default` variant of the right column in Figure 3 there are a few instances that have a high running time despite a parameter value of zero. This is an argument against the suggested regression being a "good" explanation. Also, in the bottom right plot one can see that there are no such (drastic) outliers. Hence, the correlation with the `hint` variant with the gap is considerably "better" than with the slower `default` variant (despite only small differences in the average running time, see Table 1). This is also reflected in the corresponding correlation coefficients of 0.61 and 0.35 respectively (see two rightmost columns in Table 2) and, thus, supports the correlation coefficient as reasonable measure. All in all, the results for 2-Club indicate that the correlation coefficients reveal exponential dependencies between the running time and the considered parameter.

**Correlations with number of vertices.** The results for the other problems are somewhat similar to the ones for 2-Club. The correlation coefficient for the number of vertices is highest for all problems with the `noTK` configuration, whereas with the configurations based on Turing kernels it is significantly lower (or even negative). This is somehow expected, as all our ILP formulations use $\mathcal{O}(n)$ binary variables. State-of-the-art ILP solvers are highly complex ("a bag of tricks") and able to solve instances with millions of integer variables efficiently. Thus, the correlation of around 0.5 (for `noTK`) with the number of vertices is higher than for the other variants, but not the overall highest correlations (see second row and second column in Figure 3 for the plot corresponding to the highest correlation).

**Correlations with generalized degeneracy.** The Turing kernel approaches correlate in general better with the 2/3-degeneracy, notable exceptions are the `2club_hint` and `3club_hint` variants. As expected, across all problems the highest correlations with the 2/3-degeneracy are achieved by the `full` variants: The 2-degeneracy (3-degeneracy) is in our dataset on average more than five times (more than three times) smaller than $n$. Hence, the high correlations for the `noTK` variants with $n$ translate to high correlations for the `full` variants with 2/3-degeneracy. For the `noTK` configuration there is barely any correlation with the 2/3-degeneracy. It thus seems that the ILP solver cannot exploit the 2/3-degeneracy – at least with the given ILP formulations.

**Correlations for `default` and `hint` variants.** As discussed in Section 4, the `default` and `hint` variants are considerably faster than the `full` variants due to having access to some (perfect) lower bound. As we use the black box of an ILP-solver we do not have theoretical running time bounds covering the effect of this lower bound. Also note that the running time differences between the `default` and `hint` variants are much smaller than the differences between other pairs of variants (see Table 1). However, we can observe significant differences in the correlations of the `default` and `hint` variants for 2-Club and 3-Club with respect to the generalized degeneracy and the gap parameter. Moreover, the correlation coefficients support some speculations: The correlations in the middle table of Table 2 suggest that this running time improvement is not (so much) correlated to the 2/3-degeneracy but to another parameter. For finding clubs the gap-parameter is a good explanation: `2club_hint` and `3club_hint` have high correlations with the gap parameter; this can also be seen in the bottom right $2 \times 2$ plot subgrid of Figure 3. Thus, with a better lower bound computation (i. e., some actual heuristic) we suspect the correlation of the `default` variant with the degeneracy to decrease and increase with the gap parameter.

For plexes this argumentation does not hold. There is rarely any difference in the correlation coefficients of the `default` and `hint` variant with the 2/3-degeneracy and the gap-parameter. The reason is simple: while the gap is considerably smaller than the 2/3-degeneracy for 2/3-Club (the clubs are rather large), this is not the case for 2/3-Plex (the plexes are quite small), see Figure 4 (in the appendix). Thus, for 2/3-Plex the correlations differ only marginally between the `hint` and `default` variants. Moreover, this explains very well why despite `2club_noTK` being quite slow compared to `2plex_noTK` the variant `2club_hint` is much faster than `2plex_hint`: The average gap for our 2-club instances is 7.9, hence the exponential running time dependency on the gap is manageable. For 3-club instances, the average gap is 18.6 which, apparently, is one of the reasons why the `3club` variants are much slower than the `2club` variants.

## 6 Conclusion

We provided theoretical bounds for algorithms solving $s$-Club and $s$-Plex and experimentally tested the employed Turing kernelization for $s \in \{2, 3\}$. We found that the Turing kernel approach improves the running time significantly more for clubs than plexes. We believe that this is due to the fact that the $x$-degeneracy is defined based on distance. This suggests the need for exploring notions more suitable for finding plexes.

We also discussed the correlation between the observed running times and the theoretical bounds. Yet, there is still a large gap between theory and practice: for example, the bases of the exponential function obtained by regression are all below 1.1 – much smaller than current theoretical results suggest. We are confident that our approach based on correlation coefficients can help to close this gap. The approach is easy to employ and quite flexible.

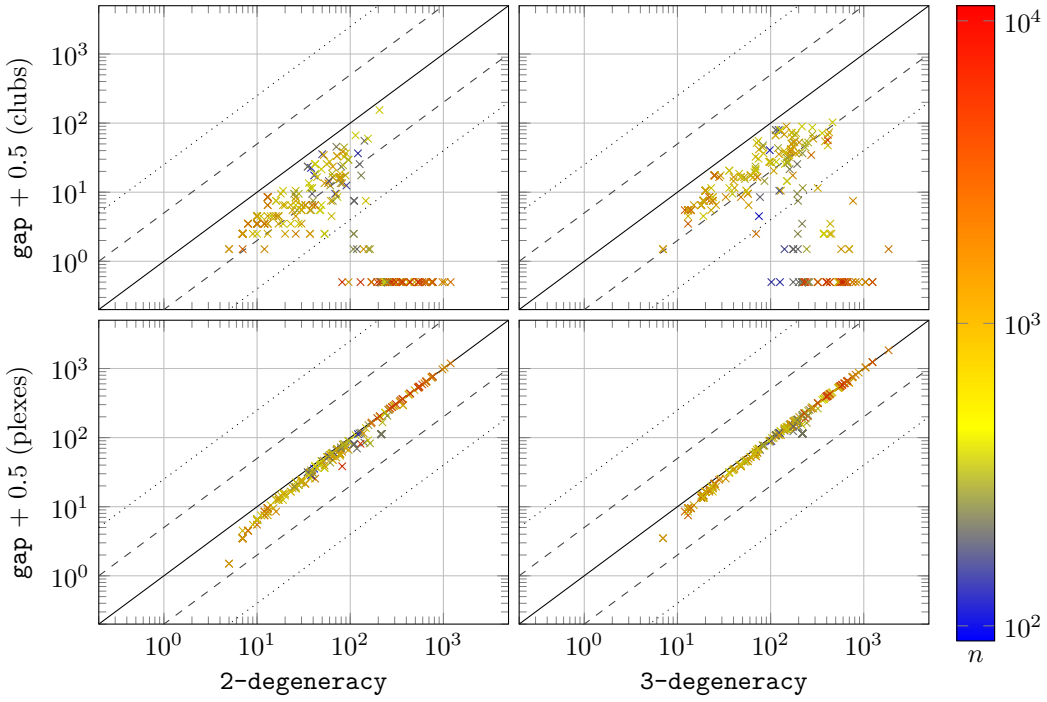



**Figure 4** Relation between 2-degeneracy, gap, and number of vertices. We added 0.5 to the gap as we use log scale and several instances have a gap of zero (for `2club` and `3club`). The diagonal lines mark factors of 1 (solid), 5 (dashed) and 25 (dotted).

Finally, we discuss some directions for future work:

- Checking whether the running times correlate with multiple parameters is an easy (but tedious) extension. The whole process should allow for relatively easy automation. An automated tool could generate a list of likely correlations from experimental results. These can then be analyzed theoretically with the parameterized complexity framework. This way, practice could give more impulses for theory.
- Our approach is not limited to analyzing running times. Other objectives could be the size of preprocessed instances (using the kernelization framework from parameterized algorithmics) or approximation factors of heuristics or approximation algorithms.
- While we use worst-case analysis, average case analysis or smoothed analysis are further natural candidates.
- Improving our general approach: For example, how to incorporate timeouts? Or are different correlation coefficients better suited?

## References

1 Maria Teresa Almeida and Filipa D. Carvalho. The k-club problem: new results for k= 3. *CIO - Centro de Investigação Operacional*, CIO - Working Paper 3/2008, 2008.

2 Maria Teresa Almeida and Filipa D. Carvalho. An analytical comparison of the lp relaxations of integer models for the k-club problem. *European Journal of Operational Research*, 232(3):489–498, 2014. doi:10.1016/j.ejor.2013.08.004.

3 Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum $k$-plex problem. *Operations Research*, 59(1):133–142, 2011. doi:10.1287/opre.1100.0851.

4 Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. An efficient branch-and-bound solver for hitting set. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '22)*, pages 209–220. SIAM, 2022. doi:10.1137/1.9781611977042.17.

5 Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. Heuristics for finding $k$-clubs in an undirected graph. *Computers & Operations Research*, 27(6):559–569, 2000.

6 Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, 138(1):21–28, 2002. doi:10.1016/S0377-2217(01)00133-3.

7 Austin Buchanan and Hosseinali Salemi. Parsimonious formulations of low-diameter clusters. *Optimization Online Eprints*, 2017.

8 Lijun Chang. Efficient maximum clique computation over large sparse graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*, pages 529–538. ACM, 2019. doi:10.1145/3292500.3330986.

9 Maw-Shang Chang, Ling-Ju Hung, Chih-Ren Lin, and Ping-Chen Su. Finding large k-clubs in undirected graphs. *Computing*, 95(9):739–758, 2013. doi:10.1007/s00607-012-0263-3.

10 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

11 Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Cliques are too strict for representing communities: Finding large $k$-plexes in real networks. In *Proceedings of the 26th Italian Symposium on Advanced Database Systems*, volume 2161 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018. URL: http://ceur-ws.org/Vol-2161/paper41.pdf.

12 Alessio Conte, Tiziano De Matteis, De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. D2K: scalable community detection in massive networks via small-diameter $k$-plexes. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*, pages 1272–1281. ACM, 2018. doi:10.1145/3219819.3220093.

13 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

14 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM J. Exp. Algorithmics*, 18, 2013. doi:10.1145/2543629.

15 Sepp Hartung, Christian Komusiewicz, and André Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. *Journal of Graph Algorithms and Applications*, 19(1):155–190, 2015. doi:10.7155/jgaa.00352.

16 Sepp Hartung, Christian Komusiewicz, André Nichterlein, and Ondrej Suchý. On structural parameterizations for the 2-club problem. *Discrete Applied Mathematics*, 185:79–92, 2015. doi:10.1016/j.dam.2014.11.026.

17 Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2010)*, volume 6140 of *Lecture Notes in Computer Science*, pages 186–202. Springer, 2010. doi:10.1007/978-3-642-13520-0_23.

18 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

19 Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. doi:10.1016/S0304-3975(01)00414-5.

20 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Computing dense and sparse subgraphs of weakly closed graphs. In *Procedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, pages 20:1–20:17, 2020. doi:10.4230/LIPIcs.ISAAC.2020.20.

21 Tomohiro Koana, Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. *ACM Journal of Experimental Algorithmics*, 26:1.3:1–1.3:30, 2021. doi:10.1145/3439801.

22 Christian Komusiewicz. Multivariate algorithmics for finding cohesive subnetworks. *Algorithms*, 9(1):21, 2016. doi:10.3390/a9010021.

23 Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. doi:10.1016/j.tcs.2009.04.021.

24 Yajun Lu, Esmaeel Moradi, and Balabhaskar Balasundaram. Correction to: Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. *Optimization Letters*, 12(8):1959–1969, 2018. doi:10.1007/s11590-018-1273-7.

25 Esmaeel Moradi and Balabhaskar Balasundaram. Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. *Optimization Letters*, 12(8):1947–1957, 2018. doi:10.1007/s11590-015-0971-7.

26 Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing wrong data without doing anything obviously wrong! In Mary Lou Soffa and Mary Jane Irwin, editors, *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS 2009)*, pages 265–276. ACM, 2009. doi:10.1145/1508244.1508275.

27 Mohammad Javad Naderi, Austin Buchanan, and Jose L. Walteros. Worst-case analysis of clique mips. *Mathematical Programming*, 195(1):517–551, 2022. doi:10.1007/s10107-021-01706-2.

28 F. Mahdavi Pajouh and B. Balasundaram. On inclusionwise maximal and maximum cardinality $k$-clubs in graphs. *Discrete Optimization*, 9:84–97, 2012.

29 Foad Mahdavi Pajouh, Esmaeel Moradi, and Balabhaskar Balasundaram. Detecting large risk-averse 2-clubs in graphs with random edge failures. *Annals of Operations Research*, 249(1-2):55–73, 2017. doi:10.1007/s10479-016-2279-0.

30 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. accessed 01.01.2022. URL: https://networkrepository.com.

31 Ryan A. Rossi, David F. Gleich, and Assefaw Hadish Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM J. Sci. Comput.*, 37(5), 2015. doi:10.1137/14100018X.

32 Hosseinali Salemi and Austin Buchanan. Parsimonious formulations for low-diameter clusters. *Mathematical Programming Computation*, 12(3):493–528, 2020. doi:10.1007/s12532-020-00175-6.

33 Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6(5):883–891, 2012. doi:10.1007/s11590-011-0311-5.

34 Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6(1):139–154, 1978.

35 Darren Strash and Louise Thompson. Effective data reduction for the vertex clique cover problem. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX '22)*, pages 41–53. SIAM, 2022. doi:10.1137/1.9781611977042.4.

36 Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56(1):113–130, 2013. doi:10.1007/s10589-013-9548-5.

37 Alexander Veremyev, Oleg A. Prokopyev, and Eduardo L. Pasiliao. Critical nodes for distance-based connectivity and related problems in graphs. *Networks*, 66, 2015. doi:10.1002/net.21622.

38 Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS J. Comput.*, 27(1):164–177, 2015. doi:10.1287/ijoc.2014.0618.

39 Jose L. Walteros and Austin Buchanan. Why is maximum clique often easy in practice? *Operations Research*, 68(6):1866–1895, 2020. doi:10.1287/opre.2019.1970.

40 Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015. doi:10.1016/j.ejor.2014.09.064.

41 Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017. doi:10.1016/j.ic.2017.06.001.

# Kernelization for Spreading Points

**Fedor V. Fomin**
University of Bergen, Norway

**Petr A. Golovach**
University of Bergen, Norway

**Tanmay Inamdar**
University of Bergen, Norway

**Saket Saurabh**
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway

**Meirav Zehavi**
Ben-Gurion University, Beer-Sheva, Israel

## Abstract

We consider the following problem about dispersing points. Given a set of points in the plane, the task is to identify whether by moving a small number of points by small distance, we can obtain an arrangement of points such that no pair of points is "close" to each other. More precisely, for a family of $n$ points, an integer $k$, and a real number $d > 0$, we ask whether at most $k$ points could be relocated, each point at distance at most $d$ from its original location, such that the distance between each pair of points is at least a fixed constant, say 1. A number of approximation algorithms for variants of this problem, under different names like distant representatives, disk dispersing, or point spreading, are known in the literature. However, to the best of our knowledge, the parameterized complexity of this problem remains widely unexplored. We make the first step in this direction by providing a kernelization algorithm that, in polynomial time, produces an equivalent instance with $\mathcal{O}(d^2k^3)$ points. As a byproduct of this result, we also design a non-trivial fixed-parameter tractable (FPT) algorithm for the problem, parameterized by $k$ and $d$. Finally, we complement the result about polynomial kernelization by showing a lower bound that rules out the existence of a kernel whose size is polynomial in $k$ alone, unless NP $\subseteq$ coNP /poly.

**2012 ACM Subject Classification** Theory of computation → Packing and covering problems; Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** parameterized algorithms, kernelization, spreading points, distant representatives, unit disk packing

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.48

**Related Version** *Full Version*: https://arxiv.org/abs/2308.07099

**Funding** *Fedor V. Fomin*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
*Petr A. Golovach*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
*Tanmay Inamdar*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
*Saket Saurabh*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416), and Swarnajayanti Fellowship (No. DST/SJF/MSA01/2017-18).
*Meirav Zehavi*: This work was supported by the European Research Council (ERC) grant titled PARAPATH.

**Acknowledgements** We thank the anonymous reviewers for their valuable comments.





## 1 Introduction

The problem of dispersing a family of objects is a common theme in many situations in computational geometry. It appears naturally in the wide range of settings that require assigning elements to locations. In many scenarios, dispersing has two often contradicting objectives. On the one hand, it is desirable not to place the objects too close to each other. This can be due to a variety of reasons, e.g., placing customers in a restaurant in socially distant manner, to placing wireless sensors far from each other in order to avoid interference. On the other hand, we may already have an existing placement of the objects, and wish to optimize the resources spent on moving the objects.

With this motivation, we consider the following mathematical model of the dispersing problems. In this model, our aim is to modify a given arrangement of points in the plane, by moving some of the points into new positions within a given distance, such that the Euclidean distance between each pair of points in the final arrangement is at least a fixed constant, say 2. Equivalently, the problem can be reformulated in terms of finding a non-overlapping arrangement of unit disks, formulated below as the problem Disk Dispersal.

Disk Dispersal

| | |
|---|---|
| *Input:* | A family $\mathcal{S}$ of $n$ unit disks, an integer $k \geq 0$, and a real $d \geq 0$. |
| *Task:* | Decide whether it is possible to obtain from $\mathcal{S}$ a family of non-overlapping unit disks $\mathcal{P}$ by moving at most $k$ disks into new positions in such a way that each unit disk is moved a distance at most $d$. [1] |

Disk Dispersal– and therefore, the problem of spreading points – is closely related to the problem of finding a system of *q-distant representatives*. This problem was introduced by Fiala, Kratochvíl, and Proskurowski [14] as a geometric extension of the classic combinatorial notion of the "systems of distinct representatives". For a set of geometric objects in a metric space and a number $q > 0$, the task is to choose one representative point from each object such that the selected points are at a distance at least $q$ from each other. For $k = n$, an instance $(\mathcal{S}, d, k)$ of Disk Dispersal can be viewed as an instance of the problem of finding a system of $q$-distance representatives by setting $q = 2$ and defining the set of geometric objects as follows: for each disk $D \in \mathcal{S}$, create a disk with the same center but with radius $d$ (instead of 1). This yields that Disk Dispersal is also NP-hard for $d = 2$ from the result of [14].

The problem of computing the distant representatives has applications in map labeling and data vizualization, where the goal is to place labels as close as possible to the specified features of the map but avoiding overlapping (thus the centers of labels are the centers of non-intersecting disks, ensuring that they are sufficiently separated) [9, 20, 21]. The problem is also related to problems of "imprecise points" [22, 23], the settings where locations of points are given with some precision. Approximation algorithms for this and related point spreading problems – where the goal is to place the specified number of points within a certain region so as to maximize the smallest pairwise distance between the points – were developed in [3, 4, 6, 10, 11, 12, 13, 19, 2, 18].

[1] All (unit) disks considered in the paper are open unless specified otherwise. In particular, two unit disks touching each other are not considered to be overlapping. Due to this simplifying assumption, we avoid the discussion about placing disks such that the distance between their boundaries is infinitesimally small.

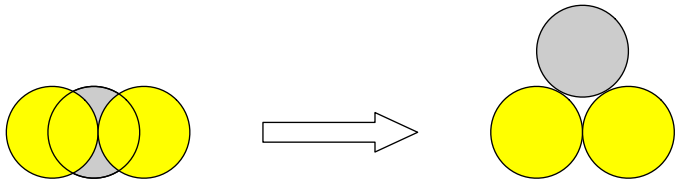


**Figure 1** An example of Disk Dispersal with $k = 1$ and $d = \sqrt{3}$. A non-overlapping arrangement of disks obtained from a family of three disks by moving the central disk at distance $\sqrt{3}$.

To the best of our knowledge, the parameterized complexity of dispersal problems are widely unexplored. The notable exception is the work of Demaine, Hajiaghayi, and Marx [7] on dispersion in graphs. In this problem, we are given an underlying edge-weighted graph, called the *connectivity graph* $G$, and a set of $k$ "agents" or "pebbles", located at a subset of vertices $G$. The task is to move the pebbles to distinct vertices and such that no two pebbles are adjacent. The movement problem is W[1]-hard parameterized by the number of pebbles, even in the case when each pebble is allowed to move at most one step.

## 1.1 Our Results

Our first result concerns kernelization (polynomial compression) of Disk Dispersal. Informally speaking, in parameterized complexity, the polynomial kernel is a polynomial-time algorithm that compresses the instance of a parameterized problem to the instance whose size is bounded by a polynomial of the parameter. Theorem 1 gives an algorithm that runs in polynomial time, and reduces the number of disks to some polynomial of $d$ and $k$.

▶ **Theorem 1.** *There is a polynomial-time algorithm that, given an instance $(\mathcal{S}, k, d)$ of Disk Dispersal, outputs an equivalent instance $(\mathcal{S}', k, d)$ of the same problem, where the number of unit disks is $|\mathcal{S}'| = \mathcal{O}((d+1)^2k^3)$, and $\mathcal{S}' \subseteq \mathcal{S}$.*

Strictly speaking, the algorithm in Theorem 1 is not a polynomial kernel according to the standard definition of this notion – we do not guarantee that the coordinates of disks, and thus the overall size of the compressed instance, is bounded by a polynomial in $k$ and $d$. We call such a compression algorithm a *partial kernel*. Further, we observe in Theorem 12 that the partial kernel from Theorem 1 can be modified to be a polynomial kernel if the centers of input disks are constrained to be rationals and we parameterize the problem by $k$, $d$, and the maximum denominator of coordinates of centers.

For a parameterized problem, given the existence of a (partial) kernel, it is usually straightforward to design a fixed-paramter tractable (FPT) algorithm by an exhaustive enumeration of all candidate solutions. For Disk Dispersal, however, this is not entirely obvious. After computing an equivalent reduced instance by applying Theorem 1, one can enumerate all possible subsets of at most $k$ unit disks that are to be moved. Now, for each such subset, we want to decide whether each unit disk in the subset can be moved by a distance of at most $d$ that results in a non-overlapping configuration. Since there are infinitely many possible target locations for each unit disk, this step requires some additional work. We show that this decision subroutine can be reduced to checking whether a system of polynomial inequalities has a solution over real numbers, which can then be determined in FPT time by using classical results from computational real algebra. Thus, we obtain the following non-trivial corollary.

▶ **Corollary 2.** *Disk Dispersal is* FPT *when parameterized by $d + k$. Specifically, it is solvable in time $(dk)^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$.*

Our next result is a companion lower bound to the partial kernelization of Theorem 1, which shows that one cannot remove the dependence on $d$ from the kernel size.

▶ **Theorem 3.** *Disk Dispersal parameterized by $k$ does not admit a polynomial kernel unless* coNP $\subseteq$ NP/poly. *This result holds even if the distance $d$ is an integer, and the centers of the given disks have rational coordinates.*

As we already mentioned, by the result of Fiala, Kratochvíl, and Proskurowski about $q$-distant representatives, Disk Dispersal is NP-hard for $d = 2$. Thus the problem is in the class para-NP for parameter $d$. However, the complexity of parameterization by $k$ is more interesting, which remains open. However, in the appendix, we show that a *rectilinear* version of Disk Dispersal is indeed W[1]-hard parameterized by $k$.

### Organization

In Section 2 we introduce basic notions. In Section 3, we consider kernelization for Disk Dispersal. Further, we give complexity lower bounds. In Section 4, we show that it is unlikely that Disk Dispersal admits a polynomial kernel when parameterized by $k$ only. Finally, in Section 5, we provide some concluding remarks and future directions.

## 2 Preliminaries

As it is common in computational geometry, we assume the *real RAM* computational model, that is, we are working with real numbers and assume that basic operations can be executed in unit time.

### Disks and Segments

For two points $A$ and $B$ in the plane, we use $AB$ to denote the line segment with endpoints at $A$ and $B$. The *distance* between $A = (x_1, y_1)$ and $B = (x_2, y_2)$ or the *length* of $AB$, is $|AB| = \|A-B\|_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. The *(open unit) disk* with a *center* $C = (c_1, c_2)$ in the plane is the set of points $(x, y)$ satisfying the inequality $(x - c_1)^2 + (y - c_2)^2 < 1$. Whenever we write "disk" we mean an *open unit disk*, unless radius or closed-ness is specified explicitly. Clearly, two disks with centers $A$ and $B$ are disjoint if and only if the distance between $A$ and $B$ is at least two. We say that the disks *touch* if $|AB| = 2$. For real numbers $a \leq b$, we use $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$ to denote a *closed interval*. For $a_1 \leq b_1$ and $a_2 \leq b_2$, $[a_1, b_1] \times [a_2, b_2] = \{(x, y) \in \mathbb{R}^2 \mid a_1 \leq x \leq b_1 \text{ and } a_2 \leq y \leq b_2\}$. A point $X$ is *properly inside* of a polygon $P$ if it is inside $P$ but $X$ is not on the boundary; if we say that $X$ is inside $P$, we allow it to be on the boundary. A disk is *(properly) iniside* of a polygon $P$ if every point of the disk is (properly) inside of $P$.

### Graphs

We use standard graph-theoretic terminology and refer to the textbook of Diestel [8] for definitions of standard notions. Let $\mathcal{S}$ be a set of geometric objects in the plane (i.e., non-empty subsets of $\mathbb{R}^2$). Then, it is possible to define an intersection graph $G(\mathcal{S})$ as follows: $G(\mathcal{S})$ contains a unique vertex corresponding to every object in $\mathcal{S}$, and there is an edge between the two vertices iff the corresponding two objects in $\mathcal{S}$ have a non-empty intersection. *Unit disk graphs* are the intersection graphs of unit disks in the plane. Note that, given a family $\mathcal{S}$ of unit disks, we can construct the corresponding unit disk graph $G(\mathcal{S})$ in quadratic time.

### Parameterized Complexity

We refer to the standard textbooks ([5, 17]) for introduction to the area and formal definitions. Here, we only give a brief overview. Let $(\mathcal{I}, k)$ be an instance of a decision problem $\Pi$, where $k$ is a non-negative integer. We say that $\Pi$ is *fixed-parameter tractable* by $k$, if there exists an algorithm that can decide whether $\mathcal{I}$ is a yes-instance of $\Pi$ in time $f(k) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$ for some computable function $f$, where $|\mathcal{I}|$ denotes the size of the instance $\mathcal{I}$. A common way to show that it is unlikely that a parameterized problem is in FPT, one can prove that it is W[1]-hard by demonstrating a *parameterized reduction* from a known W[1]-hard problem; we refer to [5] for the formal definitions of the class W[1] and parameterized reductions.

A *kernelization* (or *kernel*) for $\Pi$ is a polynomial time algorithm that, given an instance $(\mathcal{I}, k)$ of $\Pi$, outputs an equivalent instance $(\mathcal{I}', k')$ of $\Pi$ such that $|\mathcal{I}'| + k' \leq g(k)$ for a computable function $g$. A kernel is *polynomial* if $g$ is a polynomial. It can be shown that every decidable FPT problem admits a kernel. However, it is unlikely that all FPT problems have polynomial kernels. In particular, there is the now standard *cross-composition* technique to show that a parameterized problem does not admit a polynomial kernel unless NP $\subseteq$ coNP /poly.

### Systems of Polynomial Inequalities

In our FPT algorithm, we will need to find suitable locations for new disks that need to be added such that the locations are "compatible" with an existing arrangement of disks. We will achieve this by solving systems of polynomial inequalities. We use the following result.

▶ **Proposition 4** (Theorem 13.13 in [1]). *Let $R$ be a real closed field, and let $\mathcal{P} \subseteq R[X_1, \ldots, X_k]$ be a finite set of $s$ polynomials, each of degree at most $c$, and let*

$$(\exists X_1)(\exists X_2) \ldots (\exists X_k) F(X_1, X_2, \ldots, X_k)$$

*be a sentence, where $F(X_1, \ldots, X_k)$ is a quantifier-free boolean formula involving $\mathcal{P}$-atoms of type $P \odot 0$, where $\odot \in \{=, \neq, >, <\}$, and $P$ is a polynomial in $\mathcal{P}$. Then, there exists an algorithm to decide the truth of the sentence with complexity $s^{k+1}c^{\mathcal{O}(k)}$ in $D$,[2] where $D$ is the ring generated by the coefficients of the polynomials in $\mathcal{P}$.*

Furthermore, a point $(X_1^*, \ldots, X_k^*)$ satisfying $F(X_1, \ldots, X_k)$ can be computed in the same time by Algorithm 13.2 (sampling algorithm) of [1] (see Theorem 13.11 of [1]).

## 3 Kernelization and FPT Algorithms for Disk Dispersal

In this section, we first prove Theorem 1 on partial kernel for Disk Dispersal parameterized by $k+d$. Specifically, the output instance of the partial kernel is guaranteed to consist of only $\mathcal{O}(d^2k^3)$ unit disks. In case the coordinates of the disks in the input instance are rationals of the form $a + \frac{b}{c}$ where $b, c$ are bounded by a fixed constant (or a polynomial in $k + d$), our partial kernel in fact yields a (normal) kernel. Finally, using our partial kernel, we prove in Corollary 2 that Disk Dispersal is FPT parameterized by $k + d$.

The proofs of our partial kernels begin with the simple observation that if we are given a yes-instance, then the unit disk graph corresponding to the input set of unit disks admits a vertex cover of size at most $k$. So, in polynomial time we obtain a vertex cover $U$ of size at

[2] That is, the algorithm performs $s^{k+1}c^{\mathcal{O}(k)}$ operations in $D$.

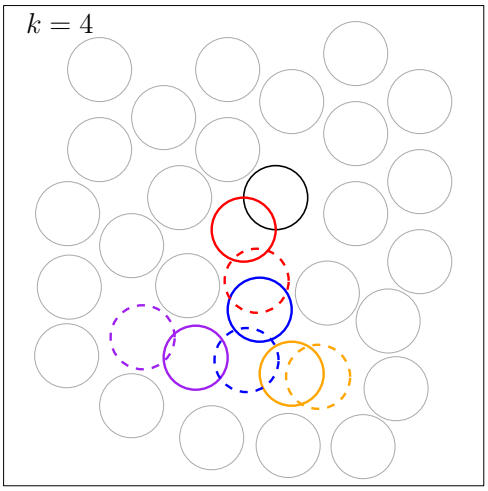



**Figure 2** Example of the propagation effect. The dotted objects correspond to a solution where an object of a certain color is replaced by the dashed object of the same color.

most $2k$. At first glance, one may think to remove all input unit disks that do not intersect any unit disk in $U$. However, we might be forced to perform movement operations that make some neighborhood sets larger (e.g., see Figure 2), which, in turn, can have a propagating effect that forces us to move unit disks that are "quite far" from all unit disks in $U$. Still, we can prove by induction on $k$ that if the input instance is a yes-instance, then it admits a solution where all the unit disks that are moved are at distance at most $\mathcal{O}(d^2k^2)$ from at least one unit disk in $U$. This gives rise to a reduction rule where we only keep the unit disks within this distance from at least one unit disk in $U$ as well as additional unit disks at some (almost negligible) distance from them.

After having reduced the number of unit disks, we can shift the unit disks that we keep so that the coordinates of their centers will be polynomial in $k + d$, under the assumption that the coordinates of the unit disks in the input instance are rationals of the form $a + \frac{b}{c}$ where $b, c$ are bounded by a fixed constant (or a polynomial in $k + d$). To obtain FPT algorithms, we first apply our partial kernels. Afterwards, we guess which disks to move. Then, we determine how to move them by solving a corresponding system of polynomial inequalities.

For the sake of formality, we will use the notion of a solution in this section as follows.

▶ **Definition 5.** *Let $(\mathcal{S}, k, d)$ be an instance of* Disk Dispersal. *A* solution *is a bijective function* $\mathsf{move}\colon \mathcal{S} \to \mathcal{P}$ *such that:*

1. *$\mathcal{P}$ is a* packing*, i.e., a non-overlapping set of unit disks.*
2. *$|\{D \in \mathcal{S} : \mathsf{move}(D) \neq D\}| \leq k$.*
3. *For every $D \in \mathcal{S}$: The distance between the centers of $D$ and $\mathsf{move}(D)$ is at most $d$.*

*We define the* set of unit disks moved by move *as $\{D \in \mathcal{S} : \mathsf{move}(D) \neq D\}$, and the* size of move *as the size of this set.*

Notice that any set of unit disks that is moved by a solution to Disk Dispersal is in particular a vertex cover (though not necessarily a minimal one) for the intersection graph of the input set of unit disks. As previously discussed, since the Vertex Cover problem admits a 2-approximation algorithm in polynomial time, this yields the following observation.

▶ **Observation 6.** *There exists a polynomial-time algorithm that, given an instance $(\mathcal{S}, k, d)$ of* Disk Dispersal*, either correctly concludes that $(\mathcal{S}, k, d)$ is a no-instance, or outputs a vertex cover of size at most $2k$ for the unit disk graph corresponding to $\mathcal{S}$.*

We will also need the following observation, which is directly implied by the fact that the area of a disk of radius $r$ is $\pi r^2$, while the area of a unit disk (whose radius is 1) is $\pi$.

▶ **Observation 7.** *The number of pairwise non-intersecting unit disks in a disk of radius $r$ is at most $r^2$.*

Towards the presentation of our partial kernel, we need to prove one lemma. Informally speaking, this lemma shows that the set of disks that may be potentially moved in a yes-instance is contained in a bounded area around a small number of disks, in particular the disks that form a vertex cover in the intersection graph. Furthermore, since all such disks, except that forming the vertex cover, are non-intersecting, this lemma eventually helps us bound the number of such disks by a polynomial in $k$ and $d$.

▶ **Lemma 8.** *Let $(\mathcal{S}, d, k)$ be a yes-instance of Disk Dispersal. Let $U$ be a vertex cover for the intersection graph of $\mathcal{S}$. Then, any minimum-sized solution to $(\mathcal{S}, k, d)$ only moves unit disks whose center is at distance at most $(d+2) \cdot k$ from the center of at least one unit disk in $U$.*

**Proof.** We prove the lemma by induction on $k$. When $k = 0$, the only minimum-sized solution to $(\mathcal{S}, k, d)$ is the one that moves no unit disk, and hence the claim trivially follows. Now, suppose that the claim holds for $k - 1 \geq 0$, and let us prove it for $k$. If the intersection graph of $\mathcal{S}$ is edgeless, then the only minimum-sized solution to $(\mathcal{S}, k, d)$ is the one that moves no unit disk, and hence the claim trivially follows as in the base case. So, we can next suppose that there exist two different unit disks $D, D' \in \mathcal{S}$ that intersect each other. See Figure 3 for an illustration.

Since $U$ is a vertex cover, it must contain at least one unit disk among $D$ and $D'$, denoted by $X$. Moreover, any solution to $(\mathcal{S}, k, d)$ must move at least one unit disk among $D$ and $D'$. Let $\mathsf{move} : \mathcal{S} \to \mathcal{P}$ be an arbitrary minimum-sized solution to $\mathcal{I} = (\mathcal{S}, k, d)$, and let $Y$ be a unit disk among $D$ and $D'$ that $\mathsf{move}$ moves to attain $\mathcal{P}$. Let $Y' = \mathsf{move}(Y)$, and let $\mathcal{S}' = (\mathcal{S} \setminus \{Y\}) \cup \{Y'\}$. We attain solution $\mathsf{move}' : \mathcal{S}' \to \mathcal{P}'$ to a new instance $\mathcal{I}' = (\mathcal{S}', k-1, d)$ as follows: for every $\tilde{D} \in \mathcal{S} \setminus \{Y\}, \mathsf{move}'(\tilde{D}) = \mathsf{move}(\tilde{D})$; $\mathsf{move}'(Y') = Y'$. Note that $\mathsf{move}'$ must be a minimum-sized solution to $(\mathcal{S}', k-1, d)$, otherwise we can obtain a solution for the original instance $(\mathcal{S}, k, d)$ that is smaller than $\mathsf{move}$, contradicting its optimality. Further, note that $(U \setminus \{Y\}) \cup \{Y'\}$ is a (not necessarily minimal) vertex cover for the intersection graph of $\mathcal{S}'$. By the inductive hypothesis, this means that $\mathsf{move}'$ only moves unit disks whose center is at distance at most $(d+2) \cdot (k-1)$ from the center of at least one unit disk in $(U \setminus \{Y\}) \cup \{Y'\}$. Moreover, the distance between the centers of $Y$ and $X$ is at most 2 (since they intersect) and the distance between the centers of $Y'$ and $Y$ is at most $d$, so the distance between the centers $Y'$ and $X$ is at most $d+2$. In turn, this means that $\mathsf{move}$ only moves unit disks at distance at most $(d+2) \cdot k$ from at least one unit disk in $U$, which concludes the proof. ◀

We are now ready to present the partial kernel for Disk Dispersal. For the reader's convenience, we restate Theorem 1 here.

▶ **Theorem 1.** *There is a polynomial-time algorithm that, given an instance $(\mathcal{S}, k, d)$ of Disk Dispersal, outputs an equivalent instance $(\mathcal{S}', k, d)$ of the same problem, where the number of unit disks is $|\mathcal{S}'| = \mathcal{O}((d+1)^2 k^3)$, and $\mathcal{S}' \subseteq \mathcal{S}$.*

**Proof.** Given an instance $(\mathcal{S}, k, d)$ of Disk Dispersal, the (partial kernel) kernelization algorithm works as follows. Based on Observation 6, it computes a vertex cover $U$ of size at most $2k$ for the intersection graph of $\mathcal{S}$. Then, it obtains $\mathcal{S}'$ from $\mathcal{S}$ by removing from $\mathcal{S}$ all

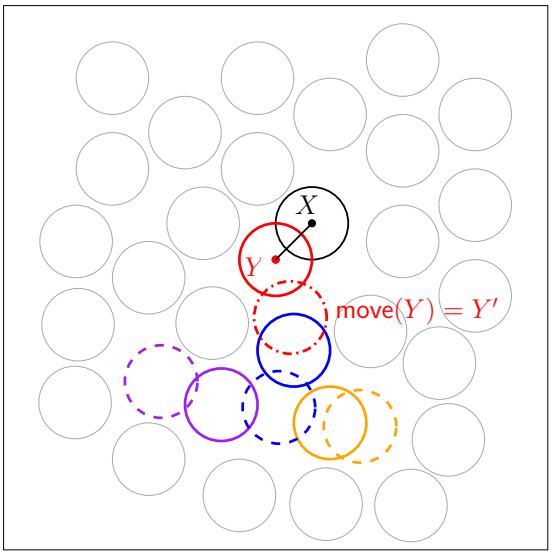



**Figure 3** Illustration for Proof of Lemma 8. A vertex cover $U$ contains disk $X$ and a solution $\mathcal{S}$ moves a disk $Y$ to its new location, $Y' = \mathsf{move}(Y)$, denoted in dash-dotted disk in red color. A new instance $\mathcal{I}'$ is obtained by replacing $Y$ with $Y'$ and reducing the budget by 1, and $U' = X \cup Y'$ is a vertex cover for the resulting intersection graph. A solution to $\mathcal{I}'$ moves the solid blue, purple, and orange disks to their new locations, shown in dashed disks of corresponding color. By inductive hypothesis, the new locations are at distance at most $(d+2)\cdot(k-1)$ from $U'$, and the distance between $X$ and $Y'$ is at most $d+2$.

the unit disks at distance more than $(d+2)\cdot(k+1)$ from all unit disks in $U$. The output instance is $(\mathcal{S}', k, d)$. Clearly, the kernelization algorithm works in polynomial time. So, it suffices to prove that $(\mathcal{S}, k, d)$ and $(\mathcal{S}', k, d)$ are equivalent and that $|\mathcal{S}'| = \mathcal{O}(d^2k^3)$.

We first prove the equivalence. In one direction, suppose that $(\mathcal{S}, k, d)$ is a yes-instance, and let $\mathsf{move} : \mathcal{S} \to \mathcal{P}$ be a solution to it. In particular, the restriction of $\mathsf{move}$ to $\mathcal{S}'$ clearly yields a packing (being a subset of $\mathcal{P}$) and moves at most as many disks as $\mathsf{move}$ does. So, the restriction of $\mathsf{move}$ to $\mathcal{S}'$ is a solution to $(\mathcal{S}', k, d)$.

In the other direction, suppose that $(\mathcal{S}', k, d)$ is a yes-instance. By Lemma 8, $(\mathcal{S}', k, d)$ admits a solution $\mathsf{move}' : \mathcal{S}' \to \mathcal{P}'$ that only moves unit disks whose centers are at distance at most $(d+2)\cdot k$ from the center of at least one unit disk in $U$.[3] Define $\mathsf{move} : \mathcal{S} \to \mathcal{P}$ for $\mathcal{P} = \mathcal{P}' \cup (\mathcal{S} \setminus \mathcal{S}')$ as follows: for every $D \in \mathcal{S}'$, $\mathsf{move}(D) = \mathsf{move}'(D)$, and for every $D \in \mathcal{S} \setminus \mathcal{S}'$, $\mathsf{move}(D) = D$. We claim that $\mathsf{move}$ is a solution to $(\mathcal{S}, k, d)$. To this end, first note that none of the unit disks in $\mathcal{P}'$ intersect each other (since $\mathsf{move}'$ is a solution to $(\mathcal{S}', k, d)$). In particular, the unit disks in $\{D \in U : \mathsf{move}(D) = D\}$ do not intersect any other unit disk in $\mathcal{P}'$. However, all unit disks in $\mathcal{S}$ that do not belong to $U$ do not intersect each other (since $U$ is a vertex cover for the intersection graph of $\mathcal{S}$). So, in $\mathcal{P}$, the only pairs of unit disks that can potentially intersect each other are pairs where one is a unit disk that was moved by $\mathsf{move}$ and the other belongs to $\mathcal{S} \setminus \mathcal{S}'$. However, the center of any unit disk $D$ that is moved by $\mathsf{move}$ is at distance at most $(d+2)\cdot k$ from the center of at least one unit disk $D'$ in $U$, and hence the center of $\mathsf{move}(D)$ is at distance at most $d + (d+2)\cdot k$ from the center of $D'$, while the center of any unit disk in $\mathcal{S} \setminus \mathcal{S}'$ is at distance more than

[3] Note that $\mathcal{S}'$ may contains unit disks whose centers are at distance larger than $(d+2)\cdot k$ (but at most $(d+2)\cdot(k+1)$) from the centers of all unit disks in $U$.

$(d+2)\cdot(k+1)$ from the centers of all unit disks in $U$. Thus, $\mathcal{P}$ cannot have a pair of unit disks that intersect each other, such that one is a unit disk that was moved by move and the other belongs to $\mathcal{S}\setminus\mathcal{S}'$. So, move is indeed a solution to $(\mathcal{S},k,d)$.

Now, note that for every $D\in U$, the unit disks whose center is at distance at most $(d+2)\cdot(k+1)$ from $D$ are contained in a disk $D'$ of radius $(d+2)\cdot(k+1)+1$ and whose center is the same as the center of $D$. So, by Observation 7 and since $U$ is a vertex cover for the intersection graph of $\mathcal{S}$, this means that there exist at most $((d+2)\cdot(k+1)+2)^2=\mathcal{O}(d^2k^2)$ unit disks in $\mathcal{S}\setminus U$ that intersect $D'$. As $|U|\leq 2k$, we conclude that $|\mathcal{S}'|\leq|U|+|U|\cdot((d+2)\cdot(k+1)+2)^2=\mathcal{O}(d^2k^3)$. ◀

To reduce the bitsize of encoding the coordinates of the unit disks in the output instance, we make use of the following lemma.

▶ **Lemma 9.** *There exists a polynomial-time algorithm that, given a set $\mathcal{D}$ of unit disks whose centers have rational coordinates, a partition $(\mathcal{D}_1,\mathcal{D}_2,\ldots,\mathcal{D}_\ell)$ of $\mathcal{D}$, and $r\in\mathbb{N}$, outputs a set $\mathcal{D}'$ of unit disks whose centers have rational coordinates and a bijective function $f:\mathcal{D}\to\mathcal{D}'$ with the following properties.*

- *For all $i\in\{1,2,\ldots,\ell\}$, $\mathcal{D}_i$ and $\{f(D):D\in\mathcal{D}_i\}$ are* isometric*, that is, for all $D,D'\in\mathcal{D}_i$, we have* $\mathsf{distance}(D,D')=\mathsf{distance}(f(D),f(D'))$.
- *For all distinct $i,j\in\{1,2,\ldots,\ell\}$, $D\in\mathcal{D}_i$ and $D'\in\mathcal{D}_j$, we have* $\mathsf{distance}(D,D')>r$.
- *Encoding the coordinates (in unary) of all the unit disks in $\{f(D):D\in\mathcal{D}_i\}$ requires space polynomial in $r,|\mathcal{D}|,m=\max\limits_{i=1}^{\ell}\max\limits_{D,D'\in\mathcal{D}_i}\mathsf{distance}(D,D')$ and $N=\max_{b,c}(b+c)$ over every $b,c\in\mathbb{N},b<c$, and $b,c$ are coprime, such that $a+\frac{b}{c}$ is a coordinate of a center of a unit disk in $\mathcal{D}$.*

**Proof.** For every $i\in\{1,2,\ldots,\ell\}$, let $L_i$ be a leftmost unit disk in $\mathcal{D}_i$ (i.e., with a smallest $x$-coordinate of its center), and let $D_i$ be a bottommost unit disk in $\mathcal{D}_i$ (i.e., with a smallest $y$-coordinate of its center), and denote their centers by $(x_i^{\mathsf{left}},y_i^{\mathsf{left}})$ and $(x_i^{\mathsf{bottom}},y_i^{\mathsf{bottom}})$, respectively. Now, for every $i\in\{1,2,\ldots,\ell\}$ and every $D\in\mathcal{D}_i$ with center $(x,y)$, define $f(D)$ as the unit disk whose center is $(x-x_i^{\mathsf{left}}+(i-1)\cdot(m+r),y-y_i^{\mathsf{bottom}}+(i-1)\cdot(m+r))$. We define $\mathcal{D}'$ as the set of unit disks assigned by $f$. Clearly, $f:\mathcal{D}\to\mathcal{D}'$ is bijective and the third property in the lemma holds.

For the first property, consider two unit disks $D,D'\in\mathcal{D}_i$ for some $i\in\{1,2,\ldots,\ell\}$ with centers $(x,y)$ and $(x',y')$, respectively. Then, $\mathsf{distance}(f(D),f(D'))$ is equal to the square root of $((x-x_i^{\mathsf{left}}+(i-1)\cdot(m+r))-(x'-x_i^{\mathsf{left}}+(i-1)\cdot(m+r)))^2+((y-y_i^{\mathsf{bottom}}+(i-1)\cdot(m+r))-(y'-y_i^{\mathsf{bottom}}+(i-1)\cdot(m+r)))^2$, which is precisely $\sqrt{(x-x')^2+(y-y')^2}=\mathsf{distance}(D,D')$. So, the first property in the lemma holds.

For the second property, consider two unit disks $D\in\mathcal{D}_i,D'\in\mathcal{D}_j$ for some $i,j\in\{1,2,\ldots,\ell\}$, $i<j$, with centers $(x,y)$ and $(x',y')$, respectively. Then, $\mathsf{distance}(f(D),f(D'))$ is equal to the square root of $((x'-x_j^{\mathsf{left}}+(j-1)\cdot(m+r))-(x-x_i^{\mathsf{left}}+(i-1)\cdot(m+r)))^2+((y'-y_j^{\mathsf{bottom}}+(j-1)\cdot(m+r))-(y-y_i^{\mathsf{bottom}}+(i-1)\cdot(m+r)))^2$. Observe that $x'\geq x_j^{\mathsf{left}},y'\geq y_j^{\mathsf{bottom}},x\leq x_i^{\mathsf{left}}+m,y\leq y_i^{\mathsf{bottom}}+m$. So, the above expression is lower bounded by

$$\sqrt{2\left((j-1)\cdot(m+r)-(m+(i-1)\cdot(m+r))\right)^2}=\sqrt{2}\cdot((j-i)(m+r)-m)\geq\sqrt{2}r.$$

In particular, $\mathsf{distance}(f(D),f(D'))>r$. So, the second property in the lemma holds. ◀

We will also need the following simple observation.

▶ **Observation 10.** *Let $\mathcal{S}$ be a set of unit disks in the Euclidean plane. Let $D \in \mathcal{S}$. Then, by moving $D$ by a distance of at most some $d \in \mathbb{N}$, $D$ cannot intersect unit disks whose centers are at distance at least $d+2$ from the original position of the center of $D$.*

Based on Lemma 9 and Observation 10, we prove the following.

▶ **Lemma 11.** *There exists a polynomial-time algorithm that, given an instance $(\mathcal{S}, k, d)$ of* Disk Dispersal *where the centers of all disks have rational coordinates, and a partition $(\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_\ell)$ of $\mathcal{S}$ such that for all $i, j \in \{1, 2, \ldots, \ell\}, D \in \mathcal{S}_i$ and $D' \in \mathcal{S}_j$, we have $\mathsf{distance}(D, D') \geq 2d+2$, outputs an equivalent instance of* Disk Dispersal*, respectively, with the same parameters $k, d$ and number of unit disks, where encoding the coordinates of all the unit disks (in unary) requires space polynomial in $d$, $|\mathcal{S}|$, $m = \max_{i=1}^{\ell} \max_{D,D' \in \mathcal{S}_i} \mathsf{distance}(D, D')$ and $N = \max_{b,c}(b+c)$ over every $b, c \in \mathbb{N}, b < c$, such that $a + \frac{b}{c}$ is a coordinate of a center of a unit disk in $\mathcal{D}$.*

**Proof.** The algorithm simply applies the algorithm in Lemma 9 with $r = 2d+2$, and obtains $f : \mathcal{S} \to \mathcal{S}'$. Then, it returns $\mathcal{D}'$. From Lemma 9, it directly follows that encoding the coordinates of all the unit disks requires space polynomial in $d, |\mathcal{D}|, m$ and $N$. Recall that for all $i, j \in \{1, 2, \ldots, \ell\}, D \in \mathcal{S}_i$ and $D' \in \mathcal{S}_j$, we have $\mathsf{distance}(D, D') \geq 2d+2$, and this property is preserved under the mapping $f$ (by our choice of $r$). So, Observation 10 implies that the sub-instances induced by the different sets $\mathcal{S}_i$ are "independent" from each other: we cannot move unit disks in one set $\mathcal{S}_i$ so that they intersect unit disks in another set $\mathcal{S}_j$. Also, the same holds for the sub-instances they are mapped to by $f$. As every sub-instance induced by some set $\mathcal{S}_i$ is equivalent to the sub-instance it is mapped to by $f$ since $\mathcal{S}_i$ and $\{f(D) : D \in \mathcal{S}_i\}$ are isometric, we conclude that $(\mathcal{S}, k, d)$ and $(\mathcal{S}', k, d)$ are equivalent. ◀

We our now ready to present our (non-partial) kernel for Disk Dispersal. In particular, if $N$ is a constant (or polynomial in $k+d$), the parameterization can be assumed to be only by $k+d$.

▶ **Theorem 12.** *Disk Dispersal, restricted to instances where the centers of all disks have rational coordinates, admits a polynomial kernel with respect to $k+d+N$, where $N = \max_{b,c}(b+c)$ over every $b, c \in \mathbb{N}, b < c$, such that $a + \frac{b}{c}$ is a coordinate of a center of a unit disk in $\mathcal{S}$.*

**Proof.** Given an instance $(\mathcal{S}, k, d)$ of Disk Dispersal, restricted to instances where the centers of all disks have rational coordinates, the kernelization algorithm works as follows. First, we call the algorithm in Theorem 1 to obtain an equivalent instance $(\mathcal{S}', k, d)$ of Disk Dispersal. Here, $k, d$ remain unchanged, and $\mathcal{S}'$ is a subset of $\mathcal{S}$. Let $\mathcal{W} = \{W_D : D \in \mathcal{S}'\}$ where $W_D$ is a disk whose center is the same as the center of $D$ and whose radius is $d+1$. Let $\mathcal{C}$ be the set of connected components of the intersection graph of $\mathcal{W}$. Let $\mathcal{P}$ be the partition of $\mathcal{S}'$ such that two unit disks in $\mathcal{S}'$ belong to the same part if and only if there exists a connected component in $\mathcal{C}$ such that both are intersected by (possibly different) disks that belong to that component. It should be clear, from the definitions of $\mathcal{S}'$ and $\mathcal{W}$, that this is indeed a partition, and that if two unit disks in $\mathcal{S}'$ belong to different parts in this partition, then the distance between their centers is larger than $2d+2$. So, the kernelization algorithm then calls the algorithm in Lemma 11 on $(\mathcal{S}', k, d)$ and $\mathcal{P}$ as the partition of $\mathcal{D}'$, and returns its output. ◀

Lastly, based on Theorem 1 and Proposition 4, we prove Corollary 2 stating that Disk Dispersal is FPT when parameterized by $d+k$. We restate the theorem here.

▶ **Corollary 2.** *Disk Dispersal is* FPT *when parameterized by* $d + k$*. Specifically, it is solvable in time* $(dk)^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$*.*

**Proof.** Given an instance $(\mathcal{S}, k, d)$ of Disk Dispersal, the algorithm first calls the algorithm in Theorem 1 to obtain (in polynomial time) an equivalent instance $(\mathcal{S}', k, d)$ of Disk Dispersal, where $\mathcal{S}' \subseteq \mathcal{S}$ is of size $\mathcal{O}(d^2k^3)$. Then, for every $\mathcal{A} \subseteq \mathcal{S}'$ of size at most $k$ such that $\mathcal{S}' \setminus \mathcal{A}$ is a packing, the algorithm tests whether it is possible to move each unit disk in $\mathcal{A}$ by a distance of at most $d$ so that, afterwards, $\mathcal{S}'$ becomes a packing. This can be done by using the algorithm in Proposition 4 to solve the following system of polynomial inequalities, which has variables $x_A, y_A$ for every $A \in \mathcal{A}$:

- For every $S \in \mathcal{S}' \setminus \mathcal{A}$ and $A \in \mathcal{A}$: $(x_A - a)^2 + (y_A - b)^2 \geq 4$, where $(a, b)$ denotes the center of $S$.
- For every distinct $A_1, A_2 \in \mathcal{A}$: $(x_{A_1} - x_{A_2})^2 + (y_{A_1} - y_{A_2})^2 \geq 4$.
- For every $A \in \mathcal{A}$, where $(a, b)$ denotes the center of $A$ in $\mathcal{S}'$: $(x_A - a)^2 + (y_A - b)^2 \leq d^2$.

The correctness of the algorithm is immediate. For its running time analysis, notice that there are only $\sum_{i=0}^{k} \binom{|\mathcal{S}'|}{i} \leq (dk)^{\mathcal{O}(k)}$ choices for $\mathcal{A}$. Further, each of the systems of polynomial equations that are solved has at most $2k$ variables, degree 2, and $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{S}'|) \leq (dk)^{\mathcal{O}(1)}$ equations. So, by Proposition 4, it is solvable in time $(dk)^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$. In turn, we conclude that the algorithm runs in time $(dk)^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$. ◀

## 4 Kernelization lower bound for Disk Dispersal

In this section, we prove Theorem 3. To this end, we show that from several instances of Disk Appending (defined below), we can construct a single instance $\mathcal{I}'$ of Disk Dispersal such that there is a solution to $\mathcal{I}'$ if and only if there is a solution to at least one of the instances of Disk Appending. The result then follows from the cross-composition technique (see [17], Chapter 17 for more details). Disk Appending is defined as follows.

Disk Appending

| | |
|---|---|
| *Input:* | A packing $\mathcal{P}$ of $n$ unit disks inside a rectangle $R$ and an integer $\kappa \geq 0$. |
| *Task:* | Decide whether there is a packing $\mathcal{P}^*$ of $n + \kappa$ unit disks inside $R$ obtained from $\mathcal{P}$ by adding $\kappa$ new disks. |

A recent result of Fomin et al. [15, 16] shows that the problem is NP-hard. In particular, they show the following result.

▶ **Proposition 13** (Corollary 2 in [15])**.** *Disk Appending is* NP*-hard. Furthermore, it remains* NP*-hard, even when restricted to instances* $(R, \mathcal{P}, \kappa)$ *of the following form.*

- *Rectangle* $R$ *is* $[0, 2a] \times [0, 2b]$ *for integers* $a, b > 0$*. It can also be assumed that* $a = b$*.*
- *A packing* $\mathcal{P}$ *of disks with their centers inside* $R$ *such that (i) for every* $i \in \{0, \ldots, a\}$*, the disks with centers* $(2i, 0)$ *and* $(2i, 2b)$ *are in* $\mathcal{P}$ *and (ii) for every* $j \in \{0, \ldots, b\}$*, the disks with centers* $(0, 2j)$ *and* $(2a, 2j)$ *are in* $\mathcal{P}$*.*

**Proof of Theorem 3.** The reader may wish to refer to Figure 4, which explains the schematics of the reduction. We consider instances $(R, \mathcal{P}, n, \kappa)$ of Disk Appending, where $R$ is an $[0, a] \times [0, a]$ square, where $a$ is an even positive integer, $\mathcal{P}$ is a packing of $n$ disks with their centers inside $R$, such that the centers of the disks are rational, and $\kappa$ is the number of disks that need to be added inside $R$, which is compatible with $\mathcal{P}$, to obtain a packing of $n + k$ disks. We also assume that for every $i \in \{1, \ldots, a/2\}$, the disks with centers $(2i - 1, 1)$, $(2i - 1, a - 1)$, $(1, 2i - 1)$ and $(a - 1, 2i - 1)$ are in $\mathcal{P}$.

For the cross-composition, we first show the polynomial equivalence relation $\mathcal{R}$, over instances $(R_i, \mathcal{P}_i, n_i, \kappa_i)$ of Disk Appending. The instances $(R_i, \mathcal{P}_i, n_i, \kappa_i)$ and $(R_j, \mathcal{P}_j, n_j, \kappa_j)$ go to the same equivalence classes if (1) the squares $R_i$ and $R_j$ have the same dimension, (2) $\mathcal{P}_i$ and $\mathcal{P}_j$ is a packing of $n_i = n_j$ disks inside $R_i$ and $R_j$ respectively with centers having rational coordinates, and (3) $\kappa_i = \kappa_j$. All the other malformed instances go into another equivalence class (see [17] for the formal requirements of the equivalence relation). Note that $\mathcal{R}$ satisfies the properties of polynomial equivalence relation, since the equivalence can be checked in polynomial time, and (2) $\mathcal{R}$ partitions the elements of $S$ into at most $(\max_{x \in S} |x|)^{\mathcal{O}(1)}$ classes in a well-formed instance, since $\kappa_i \leq n_i$ (this can be assumed w.l.o.g. by padding the instance as required, as per Proposition 13).

Now we give a cross-composition algorithm for instances belonging to the same equivalence class. For the last equivalence class of malformed instances, we output a trivial no-instance. Thus, from now on, we focus on an equivalence class $(R_1, \mathcal{P}_1, n, \kappa), \ldots, (R_t, \mathcal{P}_t, n, \kappa)$, such that $a$ is the sidelength of every square $R_1, \ldots, R_t$. We assume w.l.o.g. that $t$ is odd, and $a$ is an even integer that is at least $10\kappa$.

For every $1 \leq i \leq t$, we construct a gadget $G_i$ as follows; see Figure 4. Let $R$ be a rectangle of height 6 and width $2\kappa + 6$. Suppose the cartesian coordinates of the bottom-left corner of $G_i$ are $(0, 0)$ (note that this coordinate system is defined only for explaining the gadget structure, and should not be confused with the coordinate system in the next paragraph). Then, we place $2(k + 3)$ disks centered at points $(1, 1), (3, 1), \ldots, (2k + 5, 1)$, as well as $(1, 5), (3, 5), \ldots, (2\kappa + 5, 5)$, and 2 additional disks centered at $(1, 3)$, and $(2\kappa + 5, 3)$. These disks lie along the perimeter of the rectangle, with centers at distance 1 from the perimeter. We call these disks *surrounding disks* (shown in green). Additionally, we place $\kappa$ disks with centers at $(4, 3), (6, 3), \ldots, (2\kappa + 2, 3)$, which are termed as *interesting disks* (shown in blue). Note that this leaves a horizontal gap of 1 between the leftmost (resp. rightmost) interesting disk and the surrounding disks with center $(1, 3)$ (resp. $(2\kappa + 5, 3)$). Now, we pad the gadget horizontally by adding columns of 3 disks on both sides of the surrounding disks in a symmetric manner, such that the width of the gadget becomes exactly $a$.

Now we describe the construction of the instance of Disk Dispersal. It might be useful to refer to a schematic description shown in Figure 4. Let $d$, the distance by which a disk can be moved, be equal to $\frac{9}{4}t^2a^2$. We place the first square $R_1$ and the corresponding packing of disks $\mathcal{P}_1$ from the first instance by placing the bottom-left of $R_1$ corner at the origin $(0, 0)$. Next, we place the instances $(R_2, \mathcal{P}_2), (R_3, \mathcal{P}_3), \ldots, (R_t, \mathcal{P}_t)$ by aligning their bottom edge along the $x$-axis, and leaving a horizontal gap of $s := \sqrt{2ad}$ between the adjacent squares. Then, we place the gadgets $G_i$ directly above the rectangle $R_i$ such that the vertical distance between the top edge of $R_i$ and the top edge of $G_i$ is equal to $h := \sqrt{d^2 - a^2}$. Since the width of every gadget $G_i$ is equal to $a$ after padding, the vertical boundaries of $R_i$ and the corresponding $G_i$ are aligned. Next, we place a set $C$ of $\kappa + 2$ co-located disks such that (1) the vertical distance between the bottom edge of $G_{(t+1)/2}$ and the centers of disks in $C$ is equal to $d/2$, and (2) the centers of the disks in $C$ are aligned with the horizontal center of the gadget $G_i$. We place a rectangle tightly enclosing the instance constructed thus far, and pack all the empty spaces outside the gadgets using disks with integral coordinates on the centers (not shown in the figure). Finally, we set the budget $k$, the number of disks that can be moved, to be $(\kappa + 1) + \kappa = 2\kappa + 1$. This finishes the construction of the instance of Disk Dispersal.

The proof of the following claim follows from the careful choice of $d$, $h$ and $s$ in terms of $a$ and $t$.

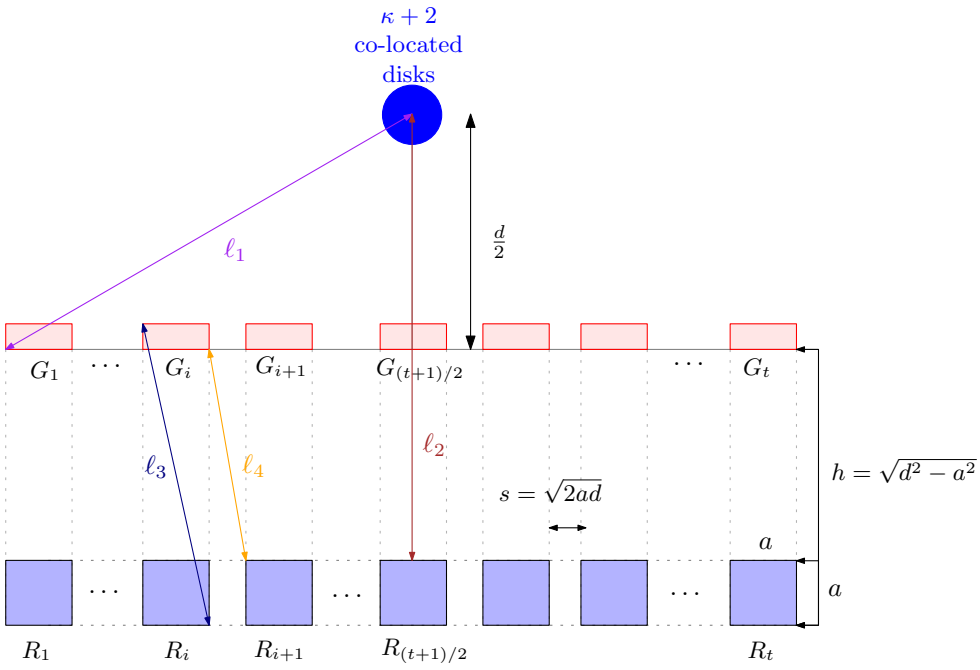



**Figure 4** Schematic depiction of an instance of Disk Dispersal obtained by OR-composition of instances $(R_i, \mathcal{P}_i, n, \kappa)$ of Disk Appending. Each instance $(R_i, \mathcal{P}_i, n, \kappa)$ is shown in a blue square of sidelength $a$. Red rectangles are gadgets $G_i$, and an example gadget is shown below. Lengths $\ell_1, \ell_2, \ell_3, \ell_4$ are defined in Claim 14, and the values of $s$, $h$, and $d$ are carefully chosen functions of $t$ and $a$, in order to ensure that $\ell_1, \ell_3 \leq d < \ell_2, \ell_4$ (note that the figure is not to scale). All the empty spaces are filled with padding disks with integral coordinates of centers. This ensures that an interesting disk from $G_i$ cannot be moved into an adjacent $R_{i\pm1}$, and thus different instances remain "isolated". In the gadget $G_i$, the *surrounding disks* are shown in green and *interesting disks* are shown in blue. Finally, purple disks are added on either side of the gadget in order to make the total width of the gadget exactly $a$. Then, the gadget $G_i$ and the corresponding square $R_i$ can be horizontally aligned as shown in the figure.

▷ Claim 14.

1. The maximum distance between the centers of disks in $C$ and any point in any $G_i$ is at most $d$ (shown as $\ell_1$ in Figure 4).
2. The minimum distance between the centers of disks in $C$ and any point in any $R_i$ is more than $d$ ($\ell_2$).
3. The maximum distance between a point in $G_i$ and a point in the corresponding $R_i$ is at most $d$ ($\ell_3$).
4. The minimum distance between a point in $G_i$ and a point in another $R_j$ is more than $d$ ($\ell_4$).

Proof. The values of $d, h$, and $s$ are chosen carefully in terms of $a$ and $t$ in order to ensure these properties. First we observe that $s = \sqrt{2ad} \geq a$, since $d = \frac{9}{4}t^2a^2$.

1. Note that the horizontal distance between the midpoint of $G_{(t+1)/2}$ and the leftmost point in $G_1$ can be upper bounded by $(t/2)(a+s) \leq (t/2)\cdot(2s) = t\cdot\sqrt{2ad}$. The vertical distance between the bottom edge of $G_{(t+1)/2}$ and the centers of disks in $C$ is $d/2$. Therefore, it suffices to show that $\left(\frac{d}{2}\right)^2 + \left(t\sqrt{2ad}\right)^2 \leq d^2$, i.e., $t^2 \cdot 2ad \leq \frac{3d^2}{4}$, i.e., $2at^2 \leq \frac{3}{4}\cdot\frac{9}{4}\cdot t^2a^2$. This holds assuming $a \geq 216$.
2. It suffices to consider the vertical distance between the centers of $C$ and the top edge of $R_{(t+1)/2}$. This vertical distance is $\frac{d}{2} + h - a$, which we want to show is greater than $d$. Note that it suffices to show that $h = \sqrt{d^2 - a^2} > \frac{d}{2}$, i.e., $a^2 < \frac{3d^2}{4}$, i.e., $\frac{243}{64}t^4a^2 > 1$. However, since $t, a \geq 1$, this is true.
3. $\ell_3^2 = h^2 + a^2 = d^2 - a^2 + a^2 = d^2$, since $h = \sqrt{d^2 - a^2}$.
4. It suffices to consider adjacent $G_i$, $R_{i+1}$ pairs (argument for $R_{i-1}$ is identical). Then, $\ell_4^2 = (h-a)^2 + s^2 = (\sqrt{d^2 - a^2} - a)^2 + 2ad = d^2 - 2a\sqrt{d^2 - a^2} + 2ad$, which we want to show is at least $d^2$. This holds since $d > \sqrt{d^2 - a^2}$. ◁

Now we explain the implications of Claim 14. In any yes-instance, at least $\kappa + 1$ disks from $C$ must be moved by a distance at most $d$. Let $C'$ be this set of disks from the set of $\kappa + 2$ co-located disks, that are moved. Note that in any gadget $G_i$, if all the $\kappa$ interesting disks are moved, then this creates an available space for placing $\kappa + 1$ disks of $C'$. On the other hand, if any set of fewer than $\kappa$ disks inducing a connected component in the *contact graph* (i.e., a special kind of intersection graph wherein there is an edge between the vertices corresponding to two disks iff their boundaries touch each other) of the disks is moved, then this creates space for at most $\kappa$ disks from $C'$ (note that the distance between $C'$ and an $R_i$ is more than $d$ by item 2 of Claim 14). However, since the budget is $2\kappa + 1$, this cannot correspond to a feasible solution. Thus, in a solution to a yes-instance, $C'$ can only be moved in the place of $\kappa$ interesting disks corresponding to a gadget $G_i$. Next, an interesting disk can be moved anywhere in the corresponding square $R_i$ (item 3), but cannot be moved to a different square $R_j$ (item 4). Then, using an argument used for the disks in $C'$, we conclude that the $k$ interesting disks can only be moved in the empty spaces in the corresponding $R_i$. Thus, the created instance of Disk Dispersal is a yes-instance iff there exists some yes-instance $(R_i, \mathcal{P}_i, n, \kappa)$ of Disk Appending. Finally, we note that Proposition 13 implies that the coordinates of the centers of the disks in each instance of Disk Dispersal can be assumed to be rational. Furthermore, by letting $s \approx \sqrt{2ad}$, and $h \approx \sqrt{d^2 - a^2}$ as rational approximations of their original values with small enough error, we can ensure that the coordinates of all the centers of the disks in the constructed instance become rational, and furthermore, the inequalities from Claim 14 continue to hold. This concludes the proof of Theorem 3. ◀

## 5 Conclusion and Open Problem

In this paper, we initiate the study of the problem of spreading points from the perspective of parameterized complexity and kernelization. We reformulate the problem in terms of moving at most $k$ unit disks by a distance of at most $d$, which we call Disk Dispersal. We design a (partial) polynomial kernel for Disk Dispersal parameterized by $k$ and $d$. Furthermore, we show that this can be transformed into a (true) kernel, assuming the coordinates of the centers of the unit disks are rational numbers with bounded denominators. We complement this result by showing that Disk Dispersal does not admit a polynomial kernel parameterized by $k$ alone, assuming coNP $\subseteq$ NP /poly. These results provide a complete picture of the kernelization complexity of Disk Dispersal.

We show that Disk Dispersal is FPT parameterized by $k+d$, by combining the (partial) kernel with a non-trivial subroutine that involves solving a system of polynomial inequalities. It is natural to ask whether the problem is fixed-parameter tractable by the individual parameters $d$ and $k$. Fiala et al. [14] have shown that Disk Dispersal is NP-hard even when $d = 2$. However, the parameterized complexity of Disk Dispersal parameterized by $k$ alone remains open. We make some preliminary progress in this direction. In the full version of the paper, we prove that the *rectilinear* version of Disk Dispersal is W[1]-hard when parameterized by $k$. This is a constrained version of Disk Dispersal, called Rectilinear Disk Dispersal, which is defined as follows.

Rectilinear Disk Dispersal

| | |
|---|---|
| *Input:* | A family $\mathcal{S}$ of $n$ unit disks, an integer $k \geq 0$, and a real $d \geq 0$. |
| *Task:* | Decide whether it is possible to obtain from $\mathcal{S}$ a family of non-overlapping disks $\mathcal{P}$ by moving at most $k$ disks into new positions parallel to the axes in such a way that each disk is moved at distance at most $d$. |

By examining our algorithmic results, namely, the (partial) kernels and the FPT algorithm parameterized by $k + d$ also hold for Rectilinear Disk Dispersal. Thus we have a complete picture of the complexity of Rectilinear Disk Dispersal, with parameters $k$ and $d$. Given this state of affairs, we conjecture that Disk Dispersal is also W[1]-hard when parameterized by $k$.

## References


1. Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*. Springer, Berlin, Heidelberg, 2009.
2. Christoph Baur and S'andor P Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.
3. Therese Biedl, Anna Lubiw, Anurag Murty Naredla, Peter Dominik Ralbovsky, and Graeme Stroud. Distant Representatives for Rectangles in the Plane. In *29th Annual European Symposium on Algorithms (ESA)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.17.
4. Sergio Cabello. Approximation algorithms for spreading points. *J. Algorithms*, 62(2):49–73, 2007. doi:10.1016/j.jalgor.2004.06.009.
5. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.
6. Erik D. Demaine, Mohammad Taghi Hajiaghayi, Hamid Mahini, Amin S. Sayedi-Roshkhar, Shayan Oveis Gharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Trans. Algorithms*, 5(3):30:1–30:30, 2009. doi:10.1145/1541885.1541891.
7. Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Dániel Marx. Minimizing movement: Fixed-parameter tractability. *ACM Trans. Algorithms*, 11(2):14:1–14:29, 2014. doi:10.1145/2650247.
8. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
9. Srinivas Doddi, Madhav V. Marathe, Andy Mirzaian, Bernard M. E. Moret, and Binhai Zhu. Map labeling and its generalizations. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 148–157. ACM/SIAM, 1997. URL: http://dl.acm.org/citation.cfm?id=314161.314250.

10 Adrian Dumitrescu and Minghui Jiang. Dispersion in unit disks. In *27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 299–310, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2010.2464.

11 Adrian Dumitrescu and Minghui Jiang. Constrained $k$-center and movement to independence. *Discret. Appl. Math.*, 159(8):859–865, 2011. doi:10.1016/j.dam.2011.01.008.

12 Adrian Dumitrescu and Minghui Jiang. Dispersion in disks. *Theory Comput. Syst.*, 51(2):125–142, 2012. doi:10.1007/s00224-011-9331-x.

13 S'andor P Fekete and Henk Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38(3):501–511, 2004.

14 Jirí Fiala, Jan Kratochvíl, and Andrzej Proskurowski. Systems of distant representatives. *Discret. Appl. Math.*, 145(2):306–316, 2005. doi:10.1016/j.dam.2004.02.018.

15 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Saket Saurabh, and Meirav Zehavi. (re)packing equal disks into rectangle. *CoRR*, abs/2211.09603, 2022. doi:10.48550/ARXIV.2211.09603.

16 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, and Meirav Zehavi. (re)packing equal disks into rectangle. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 60:1–60:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.60.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of parameterized preprocessing*. Cambridge University Press, Cambridge, 2019.

18 Tien-Ruey Hsiang, Esther M Arkin, Michael A Bender, Sandor Fekete, and Joseph SB Mitchell. Online dispersion algorithms for swarms of robots. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 382–383, 2003.

19 Tien-Ruey Hsiang, Esther M Arkin, Michael A Bender, S'andor P Fekete, and Joseph SB Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Algorithmic Foundations of Robotics V*, pages 77–93. Springer, 2004.

20 Minghui Jiang. A new approximation algorithm for labeling points with circle pairs. *Inf. Process. Lett.*, 99(4):125–129, 2006. doi:10.1016/j.ipl.2006.04.006.

21 Minghui Jiang, Jianbo Qian, Zhongping Qin, Binhai Zhu, and Robert J. Cimikowski. A simple factor-3 approximation for labeling points with circles. *Inf. Process. Lett.*, 87(2):101–105, 2003. doi:10.1016/S0020-0190(03)00256-4.

22 Maarten Löffler and Marc J. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Comput. Geom.*, 43(4):419–433, 2010. doi:10.1016/j.comgeo.2009.03.007.

23 Farnaz Sheikhi, Ali Mohades, Mark de Berg, and Ali D. Mehrabi. Separability of imprecise points. *Comput. Geom.*, 61:24–37, 2017. doi:10.1016/j.comgeo.2016.10.001.

# Lossy Kernelization for (Implicit) Hitting Set Problems

**Fedor V. Fomin**
University of Bergen, Norway

**Tien-Nam Le**
École Normale Supérieure de Lyon, France

**Daniel Lokshtanov**
University of California Santa Barbara, CA, USA

**Saket Saurabh**
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

**Stéphan Thomassé**
École Normale Supérieure de Lyon, France

**Meirav Zehavi**
Ben-Gurion University of the Negev, Beer-Sheva, Israel

## Abstract

We re-visit the complexity of polynomial time pre-processing (kernelization) for the $d$-Hitting Set problem. This is one of the most classic problems in Parameterized Complexity by itself, and, furthermore, it encompasses several other of the most well-studied problems in this field, such as Vertex Cover, Feedback Vertex Set in Tournaments (FVST) and Cluster Vertex Deletion (CVD). In fact, $d$-Hitting Set encompasses any deletion problem to a hereditary property that can be characterized by a finite set of forbidden induced subgraphs. With respect to bit size, the kernelization complexity of $d$-Hitting Set is essentially settled: there exists a kernel with $\mathcal{O}(k^d)$ bits ($\mathcal{O}(k^d)$ sets and $\mathcal{O}(k^{d-1})$ elements) and this it tight by the result of Dell and van Melkebeek [STOC 2010, JACM 2014]. Still, the question of whether there exists a kernel for $d$-Hitting Set with *fewer elements* has remained one of the most major open problems in Kernelization.

In this paper, we first show that if we allow the kernelization to be *lossy* with a qualitatively better loss than the best possible approximation ratio of polynomial time approximation algorithms, then one can obtain kernels where the number of elements is linear for every fixed $d$. Further, based on this, we present our main result: we show that there exist approximate Turing kernelizations for $d$-Hitting Set that even beat the established bit-size lower bounds for exact kernelizations – in fact, we use a *constant* number of oracle calls, each with *"near linear"* ($\mathcal{O}(k^{1+\epsilon})$) bit size, that is, almost the best one could hope for. Lastly, for two special cases of implicit 3-Hitting set, namely, FVST and CVD, we obtain the "best of both worlds" type of results – $(1+\epsilon)$-approximate kernelizations with a linear number of vertices. In terms of size, this substantially improves the exact kernels of Fomin et al. [SODA 2018, TALG 2019], with simpler arguments.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Hitting Set, Lossy Kernelization

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.49

**Related Version** *Full Version*: https://arxiv.org/abs/2308.05974

**Funding** *Fedor V. Fomin*: Research Council of Norway via the project BWCA (grant no. 314528).
*Daniel Lokshtanov*: Supported by NSF award CCF-2008838.
*Saket Saurabh*: European Research Council (ERC) grant agreement no. 819416, and Swarnajayanti Fellowship no. DST/SJF/MSA01/2017-18.
*Stéphan Thomassé*: ANR projects TWIN-WIDTH (CE48-0014-01) and DIGRAPHS (CE48-0013-01).
*Meirav Zehavi*: European Research Council (ERC) grant titled PARAPATH.





## 1 Introduction

In $d$-Hitting Set, the input consists of a universe $U$, a family $\mathcal{F}$ of sets over $U$, where each set in $\mathcal{F}$ is of size at most $d$, and an integer $k$. The task is to determine whether there exists a set $S \subseteq U$, called a *hitting set*, of size at most $k$ that has a nonempty intersection with every set of $\mathcal{F}$. The $d$-Hitting Set problem is a classical optimization problem whose computational complexity has been studied for decades from the perspectives of different algorithmic paradigms. Notably, $d$-Hitting Set is a generic problem, and hence, in particular, various computational problems can be re-cast in terms of it. Of course, Vertex Cover, the most well-studied problem in Parameterized Complexity, is the special case of $d$-Hitting Set with $d = 2$. More generally, $d$-Hitting Set encompasses a variety of (di)graph modification problems, where the task is to delete at most $k$ vertices (or edges) from a graph such that the resulting graph does not contain an induced subgraph (or a subgraph) from a family of forbidden graphs $\mathcal{F}$. Examples of some such well-studied problems include Cluster Vertex Deletion, $d$-Path Vertex Cover, $d$-Component Order Connectivity, $d$-Bounded-Degree Vertex Deletion, Split Vertex Deletion and Feedback Vertex Set in Tournaments.

Kernelization, a subfield of Parameterized Complexity, provides a mathematical framework to capture the performance of polynomial time preprocessing. It makes it possible to quantify the degree to which polynomial time algorithms succeed at reducing input instances of NP-hard problems. More formally, every instance of a parameterized problem $\Pi$ is associated with an integer $k$, which is called the *parameter*, and $\Pi$ is said to admit a *kernel* if there is a polynomial-time algorithm, called a *kernelization algorithm*, that reduces the input instance of $\Pi$ down to an equivalent instance of $\Pi$ whose size is bounded by a function $f(k)$ of $k$. (Here, two instances are equivalent if both of them are either Yes-instances or No-instances.) Such an algorithm is called an $f(k)$-*kernel* for $\Pi$. If $f(k)$ is a polynomial function of $k$, then we say that the kernel is a *polynomial kernel*. Over the last decade, Kernelization has become a central and active field of study, which stands at the forefront of Parameterized Complexity, especially with the development of complexity-theoretic lower bound tools for kernelization. These tools can be used to show that a polynomial kernel [3, 12, 18, 23], or a kernel of a specific size [9, 10, 21] for concrete problems would imply an unlikely complexity-theoretic collapse. We refer to the recent book on kernelization [17] for a detailed treatment of the area of kernelization. In this paper, we provide a number of positive results on the kernelization complexity of $d$-Hitting Set, as well as on several special cases of 3-Hitting Set.

The most well-known example of a polynomial kernel, which, to the best of our knowledge, is taught in the first class/chapter on kernelization of any course/book that considers this subject, is the classic kernel for Vertex Cover (2-Hitting Set) that is based on Buss rule. More generally, one of the most well-known examples of a polynomial kernel is a kernel with $\mathcal{O}(k^d)$ sets and elements for $d$-Hitting Set (when $d$ is a fixed constant) using the Erdös-Rado Sunflower lemma.[1] Complementing this positive result, originally in 2010, a celebrated result by Dell and van Melkebeek [10] showed that unless co-NP $\subseteq$ NP/poly, for any $d \geq 2$ and any $\epsilon > 0$, $d$-Hitting Set does not admit a kernel with $\mathcal{O}(k^{d-\epsilon})$ sets. Hence, the kernel with $\mathcal{O}(k^d)$ sets is essentially tight with respect to size. However, when it comes to the bound on the number of elements in a kernel, the situation is unclear. Abu-Khzam [1]

[1] The origins of this result are unclear. The first kernel with $\mathcal{O}(k^d)$ sets appeared in 2004 [13], but the authors do not make use of the Sunflower Lemma. To the best of our knowledge, the first exposition of the kernel based on the Sunflower Lemma appears in the book of Flum and Grohe [15].

showed that $d$-Hitting Set admits a kernel with at most $(2d-1)k^{d-1}+k$ elements. However, we do not know whether this bound is tight or even close to that. As it was written in [17, page 470]:

> *Could it be that $d$-Hitting Set admits a kernel with a polynomial in $k$ number of elements, where the degree of the polynomial does not depend on $d$? This does not look like a plausible conjecture, but we do not know how to refute it either.*

The origins of this question can be traced back to the open problems from WorKer 2010 [4, page 4]. Moreover, in the list of open problems from WorKer 2013 and FPT School 2014 [7, page 4], the authors asked whether $d$-Hitting Set admits a kernel with $f(d) \cdot k$ elements for some function $f$ of $d$ only. After being explicitly stated at these venues, this question and its variants have been re-stated in a considerable number of papers (see, e.g., [11, 17, 29, 2]), and is being repeatedly asked in annual meetings centered around parameterized complexity. Arguably, this question has become the most major and longstanding open problem in kernelization for a specific problem. In spite of many attempts, even for $d = 3$, the question whether $d$-Hitting Set admits a kernel with $\mathcal{O}(k^{2-\varepsilon})$ elements, for some $\epsilon > 0$, has still remained open.

From an approximation perspective, the optimization version of $d$-Hitting Set admits a trivial $d$-approximation. Up to the Unique Game Conjecture, this bound is tight – for any $\varepsilon > 0$, $d$-Hitting Set does not admit a polynomial time $(d-\varepsilon)$-approximation [22]. So, at this front, the problem is essentially resolved.

With respect to kernelization, firstly, the barrier in terms of number of sets, and secondly, the lack of progress in terms of the number of elements, coupled with the likely impossibility of $(d-\varepsilon)$-approximation of $d$-Hitting Set, bring lossy kernelization as a natural tool for further exploring of the complexity of this fundamental problem. We postpone the formal definition of lossy kernelization to Section 2. Informally, a polynomial size $\alpha$-approximate kernel consists of two polynomial-time procedures. The first is a pre-processing algorithm that takes as input an instance $(I, k)$ to a parameterized problem, and outputs another instance $(I', k')$ to the same problem, such that $|I'| + k' \leq k^{\mathcal{O}(1)}$. The second transforms, for every $c \geq 1$, a $c$-approximate solution $S'$ to the pre-processed instance $(I', k')$ into a $(c \cdot \alpha)$-approximate solution $S$ to the original instance $(I, k)$. Then, the main question(s) that we address in this paper is:

> Is it possible to obtain a lossy kernel for $d$-Hitting Set with a qualitatively better loss than $d$ and with $\mathcal{O}(k^{d-1-\varepsilon})$ bit-size, or at least with $\mathcal{O}(k^{d-1-\varepsilon})$ elements?

In this paper, we present a surprising answer: *not only the number of elements can be bounded by $\mathcal{O}(k)$ (rather than just $\mathcal{O}(k^{d-1-\varepsilon})$), but even the bit-size can "almost" be bounded by $\mathcal{O}(k)$!* From the perspective of the size of the kernel, this is essentially the best that one could have hoped for. Still, we only slightly (though non-negligibly) improve on the approximation ratio $d$. For example, for $d = 2$ (Vertex Cover), we attain an approximation ratio of 1.721. So, while we make a critical step that is also the first – in particular, we show that, conceptually, the combination of kernelization and approximation breaks their independent barriers – we also open up the door for further research of this kind, on this problem as well as other problems.

More precisely, we present the following results and concept. We remark that for all of our results, we use an interesting fact about the natural Linear Programming (LP) relaxation of $d$-Hitting Set: the support of any optimal LP solution to the LP-relaxation of $d$-Hitting Set is of size at most $d \cdot \mathsf{frac}$ where $\mathsf{frac}$ is the optimum (minimum value) of the

LP [20]. Furthermore, to reduce bit-size rather than only element number, we introduce an "adaptive sampling strategy" that is, to the best of our knowledge, also novel in parameterized complexity. We believe that these ideas will find further applications in kernelization in the future. More information on our methods can be found in the next section.

- **Starting Point: Linear-Element Lossy Kernel for** $d$-Hitting Set**.** First, we show that $d$-Hitting Set admits a $(d - \frac{d-1}{d})$-approximate $d \cdot \mathsf{opt}$-element kernel, where $\mathsf{opt} \leq k$ is the (unknown) optimum (that is, size of smallest solution).[2] For example, when $d = 3$, the approximation ratio is $d - \frac{d-1}{d} = 2\frac{1}{3}$, which is a notable improvement over 3. When $d = 2$, this result encompasses the classic (exact) $2 \cdot \mathsf{opt}$-vertex kernel for Vertex Cover [6, 27]. We also remark that our linear-element lossy kernel for $d$-Hitting Set is a critical component (used as a black box) in all of our other results.
- **Conceptual Contribution: Lossy Kernelization Protocols.** We extend the notions of lossy kernelization and kernelization protocols[3] to *lossy kernelization protocols*. Roughly speaking, an $\alpha$-approximate kernelization protocol can perform a bounded in $k$ number of calls (called *rounds*) to an oracle that solves the problem on instances of size (called *call size*) bounded in $k$, and besides that it runs in polynomial time. Ideally, the number of calls is bounded by a fixed constant, in which case the protocol is called *pure*. Then, if the oracle outputs $c$-approximate solutions to the instances it is given, the protocol should output a $(c \cdot \alpha)$-approximate solution to the input instance. In particular, a lossy kernel is the special case of a lossy protocol with one oracle call. The *volume* of a lossy kernelization protocol is the sum of the sizes of the calls it performs.
- **Main Contribution: Near-Linear Volume and Pure Lossy Kernelization Protocol for** $d$-Hitting Set**.** We remark that the work of Dell and van Melkebeek [10] further asserts that also the existence of an exact (i.e., 1 approximate in our terms) kernelization protocol for $d$-Hitting Set of volume $\mathcal{O}(k^{d-\epsilon})$ is impossible unless $\mathsf{co\text{-}NP} \subseteq \mathsf{NP}/\operatorname{poly}$.
  First, we show that Vertex Cover admits a (randomized) 1.721-approximate kernelization protocol of 2 rounds and call size $\mathcal{O}(k^{1.5})$. This special case is of major interest in itself: Vertex Cover is the most well-studied problem in Parameterized Complexity, and, until now, no result that breaks both bit-size and approximation ratio barriers simultaneously has been known.
  Then, we build upon the ideas exemplified for the case of Vertex Cover to significantly generalize the result: while Vertex Cover corresponds to $d = 2$, we are able to capture *all* choices of $d$. Thereby, we prove our main result: for any $\epsilon > 0$, $d$-Hitting Set admits a (randomized) pure $(d - \delta)$-approximate kernelization protocol of call size $\mathcal{O}(k^{1+\epsilon})$. Here, the number of rounds and $\delta$ are fixed constants that depend only on $d$ and $\epsilon$. While the improvement over the barrier of $d$ in terms of approximation is minor (though still notable when $d = 2$), it is a *proof of concept* – that is, it asserts that $d$ is not an impassable barrier.[4] Moreover, it does so with almost the best possible (being almost linear) output size.
- **Outlook: Relation to Ruzsa-Szemerédi Graphs.** Lastly, we present a connection between the possible existence of a $(1+\epsilon)$-approximate kernelization protocol for Vertex Cover of call size $\mathcal{O}(k^{1.5})$ and volume $\mathcal{O}(k^{1.5+o(1)})$ and a known open problem about Ruzsa-Szemerédi graphs. We discuss this result in more detail in Section 3.

[2] In fact, when the parameter is $k$, we show that the bound is better.
[3] We remark that kernelization protocols are a highly restricted special case of Turing kernels, that yet generalizes kernels.
[4] Possibly, building upon our work, further improvements on the approximation factor (though perhaps at the cost of an increase in the output size) may follow.

**Kernels for Implicit 3-Hitting Set Problems.** Lastly, we provide better lossy kernels for two well-studied graph problems, namely, Cluster Vertex Deletion and Feedback Vertex Set in Tournaments, which are known to be implicit 3-Hitting Set problems [8]. Notably, both our algorithms are based on some of the ideas and concepts that are part of our previous results, and, furthermore, we believe that the approach underlying the parts common to both these algorithms may be useful when dealing also with other hitting and packing problems of constant-sized objects. In the Cluster Vertex Deletion problem, we are given a graph $G$ and an integer $k$. The task is to decide whether there exists a set $S$ of at most $k$ vertices of $G$ such that $G - S$ is a cluster graph. Here, a cluster graph is a graph where every connected component is a clique. It is known that this problem can be formulated as a 3-Hitting Set problem where the family $\mathcal{F}$ contains the vertex sets of all *induced* $P_3$*'s* of $G$. (An induced $P_3$ is a path on three vertices where the first and last vertices are non-adjacent in $G$.) In the Feedback Vertex Set in Tournaments problem, we are given a tournament $G$ and an integer $k$. The task is to decide whether there is a set $S$ of $k$ vertices such that each directed cycle of $G$ contains a member of $S$ (i.e., $G - S$ is acyclic). It is known that Feedback Vertex Set in Tournaments can be formulated as a 3-Hitting Set problem as well, where the family $\mathcal{F}$ contains the vertex sets of all directed cycles on three vertices (triangles) of $G$.

In [16], it was shown that Feedback Vertex Set in Tournaments and Cluster Vertex Deletion admit kernels with $\mathcal{O}(k^{\frac{3}{2}})$ vertices and $\mathcal{O}(k^{\frac{5}{3}})$ vertices, respectively. This answered an open question from WorKer 2010 [4, page 4], regarding the existence of kernels with $\mathcal{O}(k^{2-\epsilon})$ vertices for these problems. The question of the existence of linear-vertex kernels for these problems is open. In the realm of approximation algorithms, for Feedback Vertex Set in Tournaments, Cai , Deng and Zang [5] gave a factor 2.5 approximation algorithm, which was later improved to 7/3 by Mnich, Williams and Végh [26]. Recently, Lokshtanov, Misra, Mukherjee, Panolan, Philip and Saurabh [24] gave a 2-approximation algorithm for Feedback Vertex Set in Tournaments. For Cluster Vertex Deletion, You, Wang and Cao [29] gave a factor 2.5 approximation algorithm, which later was improved to 7/3 by Fiorini, Joret and Schaudt [14]. It is open whether Cluster Vertex Deletion admits a 2-approximation algorithm. We remark that both problems admit approximation-preserving reductions from Vertex Cover, and hence they too do not admit $(2-\epsilon)$-approximation algorithms up to the Unique Games Conjecture.

We provide the following results for Feedback Vertex Set in Tournaments and Cluster Vertex Deletion.

- **Cluster Vertex Deletion.** For any $0 < \epsilon < 1$, the Cluster Vertex Deletion problem admits a $(1+\epsilon)$-approximate $\mathcal{O}(\frac{1}{\epsilon} \cdot \mathsf{opt})$-vertex kernel.
- **Feedback Vertex Set in Tournaments.** For any $0 < \epsilon < 1$, the Feedback Vertex Set in Tournaments problem admits a $(1+\epsilon)$-approximate $\mathcal{O}(\frac{1}{\epsilon} \cdot \mathsf{opt})$-vertex kernel.

**Reading Guide.** First, in Section 2, we present basic terminology regarding lossy kernelization. Due to lack of space, we omit the formal proofs and technical details. Instead, in Section 3, we present an overview of our proofs Afterwards, in Section 4, we conclude the extended abstract with some open problems.

## 2 Lossy Kernelization: Algorithms and Protocols

**Lossy Kernelization Algorithms.** We follow the framework of lossy kernelization presented in [25]. Here, we deal only with minimization problems where the value of a solution is its size, and where the computation of an arbitrary solution (where no optimization is enforced)

is trivial. Thus, for the sake of clarity of presentation, we only formulate the definitions for this context, and remark that the definitions can be extended to the more general setting in the straightforward way (for more information, see [25]). To present the definitions, consider a parameterized problem $\Pi$. Given an instance $I$ of $\Pi$ with parameter $k = \kappa(I)$, denote: if $k$ is a structural parameter, then $\pi_I(\mathsf{opt}) = \mathsf{opt}$, and otherwise (if $k$ is a bound on the solution size given as part of the input) $\pi_I(\mathsf{opt}) = \min\{\mathsf{opt}, k+1\}$. Moreover, for any solution $S$ to $I$, denote: if $k$ is a structural parameter, then $\pi_I(S) = |S|$, and otherwise $\pi_I(S) = \min\{|S|, k+1\}$. We remark that when $\pi$ is irrelevant (e.g., when the parameter is structural), we will drop it. A discussion of the motivation behind this definition of $\pi_I$ can be found in [25]; here, we only briefly note that it signifies that we "care" only for solutions of size at most $k$ – all other solutions are considered equally bad, treated as having size $k+1$.

▶ **Definition 1.** *Let $\Pi$ be a parameterized minimization problem. Let $\alpha \geq 1$. An* $\alpha$-approximate kernelization algorithm *for $\Pi$ consists of two polynomial-time procedures:* **reduce** *and* **lift**. *Given an instance $I$ of $\Pi$ with parameter $k = \kappa(I)$,* **reduce** *outputs another instance $I'$ of $\Pi$ with parameter $k' = \kappa(I')$ such that $|I'| \leq f(k,\alpha)$ and $k' \leq k$. Given $I, I'$ and a solution $S'$ to $I'$,* **lift** *outputs a solution $S$ to $I$ such that $\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \alpha \frac{\pi_{I'}(S')}{\pi_{I'}(\mathsf{opt}(I'))}$. If $\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \max\{\alpha, \frac{\pi_{I'}(S')}{\pi_{I'}(\mathsf{opt}(I'))}\}$ holds, then the algorithm is termed* strict.

In case $\Pi$ admits an $\alpha$-approximate kernelization algorithm where the output has size $f(k,\alpha)$, or where the output has $g(k,\alpha)$ "elements" (e.g., vertices), we say that $\Pi$ admits an $\alpha$-approximate kernel of size $f(k,\alpha)$, or an $\alpha$-approximate $g(k,\alpha)$-element kernel, respectively. When it is clear from context, we simply write $f(k)$ and $g(k)$. When it is guaranteed that $|I'| \leq f(k',\alpha)$ rather than only $|I'| \leq f(k,\alpha)$, then we say that the lossy kernel is *output-parameter sensitive*.

We only deal with problems that have constant-factor polynomial-time approximation algorithms, and where we may directly work with (the unknown) $\mathsf{opt}$ as the parameter (then, $\pi$ can be dropped). However, working with $k$ (and hence $\pi$) has the effect of artificially altering kernel sizes, but not so if one remembers that $k$ and $\mathsf{opt}$ are different parameterizations. The following lemma clarifies a relation between these two parameterizations.

▶ **Lemma 2.** *Let $\Pi$ be a minimization problem that, when parameterized by the optimum, admits an $\alpha$-approximate kernelization algorithm $\mathfrak{A}$ of size $f(\mathsf{opt})$ (resp., an $\alpha$-approximate $g(\mathsf{opt})$-element kernel). Then, when parameterized by $k$, a bound on the solution size that is part of the input, it admits an $\alpha$-approximate kernelization algorithm $\mathfrak{B}$ of size $f(\frac{k+1}{\alpha})$ (resp., an $\alpha$-approximate $g(\frac{k+1}{\alpha})$-element kernel).*

**Proof.** We design $\mathfrak{B}$ as follows. Given an instance $(I,k)$ of $\Pi$, **reduce** of $\mathfrak{B}$ calls **reduce** of $\mathfrak{A}$ on $I$. If the output instance size is at most $f(\frac{k+1}{\alpha})$ (resp., the output has at most $g(\frac{k+1}{\alpha})$ elements), then it outputs this instance with parameter $k' = k$. Otherwise, it outputs a trivial constant-sized instance. Given $(I,k), (I',k')$ and a solution $S'$ to $(I',k')$, if $I'$ is the output of the **reduce** procedure of $\mathfrak{A}$ on $I$, then **lift** of $\mathfrak{B}$ calls **lift** of $\mathfrak{A}$ on $I, I', S'$ and outputs the result. Otherwise, it outputs a trivial solution to $I$.

The **reduce** and **lift** procedures of $\mathfrak{B}$ clearly have polynomial time complexities, and the definition of $\mathfrak{B}$ implies the required size (or element) bound on the output of **reduce**. It remains to prove that the approximation ratio is $\alpha$. To this end, consider an input $(I,k), (I',k'), S'$ to **lift** of $\mathfrak{B}$. Let $S$ be its output. We differentiate between two cases.

- First, suppose that $\mathsf{opt}(I) \geq \frac{k+1}{\alpha}$. Then, $\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \frac{k+1}{\frac{k+1}{\alpha}} = \alpha \leq \alpha \frac{\pi_{I'}(S')}{\pi_{I'}(\mathsf{opt}(I'))}$ (where the last inequality follows because $|S'| \geq \mathsf{opt}(I')$ and hence $\pi_{I'}(S') \geq \pi_{I'}(\mathsf{opt}(I'))$).

- Second, suppose that $\mathsf{opt}(I) < \frac{k+1}{\alpha}$. Then, it necessarily holds that $I'$ is the output of the **reduce** procedure of $\mathfrak{A}$ on $I$. Moreover, note that $\mathsf{opt}(I') \leq \mathsf{opt}(I)$ and $k' = k$. So, if $|S'| \geq k' + 2$, then $\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \frac{k+1}{\pi_I(\mathsf{opt}(I))} = \frac{k'+1}{\mathsf{opt}(I)} \leq \frac{k'+1}{\mathsf{opt}(I')} = \frac{\pi_{I'}(S')}{\pi_{I'}(\mathsf{opt}(I'))}$. Else, we suppose that $|S'| \leq k' + 1$ and hence $\pi_{I'}(S') = |S'|$. Then,

$$\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \frac{|S|}{\pi_I(\mathsf{opt}(I))} = \frac{|S|}{\mathsf{opt}(I)} \leq \alpha \frac{|S'|}{\mathsf{opt}(I')} = \alpha \frac{\pi_{I'}(S')}{\pi_{I'}(\mathsf{opt}(I'))}.$$

Here, the second inequality follows because the approximation ratio of $\mathfrak{A}$ is $\alpha$.
This completes the proof. ◀

Approximate kernelization algorithm often use strict reduction rules, defined as follows.

▶ **Definition 3.** *Let $\Pi$ be a parameterized minimization problem. Let $\alpha \geq 1$. An $\alpha$-strict reduction rule for $\Pi$ consists of two polynomial-time procedures:* **reduce** *and* **lift**. *Given an instance $I$ of $\Pi$ with parameter $k = \kappa(I)$,* **reduce** *outputs another instance $I'$ of $\Pi$ with parameter $k' = \kappa(I') \leq k$. Given $I, I'$ and a solution $S'$ to $I'$,* **lift** *outputs a solution $S$ to $I$ such that $\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \max\{\alpha, \frac{\pi_{I'}(S')}{\pi_{I'}(\mathsf{opt}(I'))}\}$.*

▶ **Proposition 4** ([25]). *Let $\Pi$ be a parameterized problem. For any $\alpha \geq 1$, an approximate kernelization algorithm for $\Pi$ that consists only of $\alpha$-strict reduction rules has approximation ratio $\alpha$. Furthermore, it is strict.*

**Lossy Kernelization Protocols.** We extend the notion of lossy kernelization algorithms to lossy kernelization protocols as follows.

▶ **Definition 5** (Lossy Kernelization Protocol). *Let $\Pi$ be a parameterized minimization problem with parameter $k$. Let $\alpha \geq 1$. An* $\alpha$-approximate kernelization protocol *of* call size $f(k, \alpha)$ *and* $g(k, \alpha)$ rounds *for $\Pi$ is defined as follows. First, the protocol assumes to have access to an oracle $\mathfrak{O}$ that, given an instance $I'$ of $\Pi$ of size at most $f(k, \alpha)$, returns a solution $S'$ to $I'$ such that $\pi_{I'}(S') \leq \beta \pi_{I'}(\mathsf{opt}(I'))$ for minimization and $\pi_{I'}(S') \geq \frac{1}{\beta}\pi_{I'}(\mathsf{opt}(I'))$ for maximization, for some fixed $\beta > 0$. Second, for the same fixed $\beta > 0$, given an instance $I$ of $\Pi$, the protocol may perform $g(k, \alpha)$ calls to $\mathfrak{O}$ and other operations in polynomial time, and then output a solution $S$ to $I$ such that $\frac{\pi_I(S)}{\pi_I(\mathsf{opt}(I))} \leq \alpha\beta$.*

*The* volume *(or* size*) of the protocol is $f(k, \alpha)g(k, \alpha)$. In case $g(k, \alpha) = g(\alpha)$ (i.e., $g$ depends only on $\alpha$), the protocol is called* pure.

Notice that an $\alpha$-approximate kernelization algorithm is the special case of an $\alpha$-approximate kernelization protocol when the number of rounds is 1.

Practically, we think that (lossy) kernelization protocols can often be as useful as standard (lossy) kernels, and, in some cases, more useful. Like standard (lossy) kernels, they reduce the total size of what we need to solve, only that now what we need to solve is split into several instances, to be solved one after another. On the one hand, this relaxation seems to, in most cases, not be restrictive (as what we really care about is the total size of what we need to solve). On the other hand, it might be helpful if by using this relaxation one can achieve better bounds than what is known (or, even, what is possible) on the sizes of the reduced instances, or to simplify the algorithm. For example, for the case of $d$-Hitting Set, we do not know how to beat $\mathcal{O}(k^d)$ using a lossy kernel rather than a protocol.

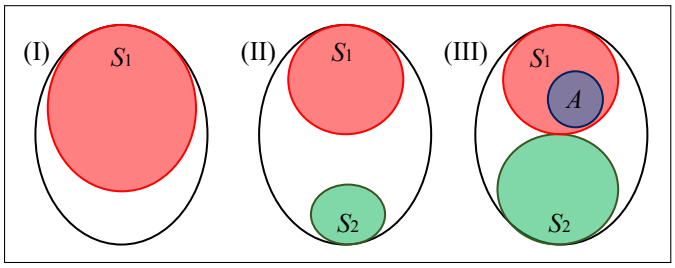



**Figure 1** The three cases encountered by our 2-call lossy kernelization protocol for Vertex Cover: (I) $|S_1|$ is large, and we return $V(G)$; (II) $|S_1|$ is small and $|S_2|$ is small, and we return $S_1 \cup S_2$; (III) $|S_1|$ is small and $|S_2|$ is large, and we return $(V(G) \setminus S_1) \cup A$.

## 3 Overview of Our Proof Ideas

In this section, we present a high-level overview of our proof ideas.

### 3.1 Linear-Element Lossy Kernel for $d$-Hitting Set

We make use of a known result about the natural LP relaxation of $d$-Hitting Set: the support of any optimal LP solution to the LP-relaxation of $d$-Hitting Set is of size at most $d \cdot \mathsf{frac}$ where $\mathsf{frac}$ is the optimum (minimum value) of the LP [20]. For the sake of completeness, we provide a proof. We then provide a lossy reduction rule that computes an optimal LP solution, and deletes all vertices assigned values at least $\frac{1}{d-1}$. Having applied this rule exhaustively, we arrive at an instance having an optimal LP solution that assigns only values strictly smaller than $\frac{1}{d-1}$. Then, it can be shown that all hitting sets are contained within the support of this LP solution. In turn, in light of the aforementioned known result, this yields an approximate $d \cdot \mathsf{frac}$-element and $(d\mathsf{frac})^d$-set kernel that is output-parameter sensitive.

The analysis that the approximation factor is $d - \frac{d-1}{d}$ is slightly more involved, and is based on case distinction. In case the number of vertices deleted is "small enough", the cost of adding them is "small enough" as well. In the more difficult case where the number of vertices deleted is "large", by making use of the already established bound on the output size as well as the drop in the fractional optimum, we are able to show that, in fact, we return a solution of approximation factor $d - \frac{d-1}{d}$ irrespective of the approximation ratio of the solution we are given. More generally, the definition of "small enough" and "large" gives rise to a trade-off that is critical for our kernelization protocol for $d$-Hitting Set, which in particular yields that we can either obtain a *negligible additive error* or directly a solution of the desired (which is some fixed constant better than $d$ but worse than $d - \frac{d-1}{d}$) approximation ratio. Specifically, this means that it is "safe" to compose our element kernel as part of other kernelization algorithms or protocols.

### 3.2 2-Round $\mathcal{O}(\mathsf{frac}^{1.5})$-Call Size Lossy Kernelization Protocol for Vertex Cover

Towards the presentation of our near-linear call size lossy kernelization protocol for $d$-Hitting Set, we abstract some of the ideas using a simpler 2-round $\mathcal{O}(\mathsf{frac}^{1.5})$-call size 1.721-approximate kernelization protocol for Vertex Cover (where $\mathsf{frac} \leq \mathsf{opt} \leq k$ is the optimum of the natural LP relaxation of Vertex Cover). First, we apply an (exact) kernelization algorithm to have a graph $G$ on at most $2\mathsf{frac}$ vertices. The purpose of having

only 2frac vertices is twofold. First, it means that to obtain a "good enough" approximate solution, it suffices that we do not pick a "large enough" (linear fraction) of vertices of $G$ to our solution. Second, it is required for a probability bound derived using union bound over vertex subsets to hold. Then, roughly speaking, the utility of the first oracle call is mainly, indirectly, to uncover a "large" (linear in $n = |V(G)|$) induced subgraph of $G$ that is "sparse", and hence can be sent to the second oracle call to be solved optimally.

More precisely, after applying the initial kernelization, we begin by sampling roughly $\mathsf{frac}^{1.5}$ edges from $G$. Then, we call the oracle on the sampled graph to obtain a solution $S_1$ to it (but not to $G$). In case that solution $S_1$ is "large" compared to the size of the vertex set of $G$ (that is, sufficiently larger than $n/2 \leq \mathsf{frac}$), we can just return the entire vertex set of $G$ (see Fig. 1). Else, we know that the subgraph of the sampled graph that is induced by $V(G) \setminus S_1$ is edgeless. In addition, we can show (due to the initial kernelization) that with high probability, every set of edges of size (roughly) at least $\mathsf{frac}^{1.5}$ that is the edge set of some induced subgraph of $G$ has been hit by our edge sample. Together, this implies that the subgraph of $G$ induced by $V(G) \setminus S_1$ has at most $\mathsf{frac}^{1.5}$ edges, and hence can be solved optimally by a second oracle call. Then, because we know that this subgraph is large compared to $G$ (else $S_1$ is large), if the oracle returned a "small" solution $S_2$ to it, we may just take this solution together with $S_1$ (which will form a vertex cover), and yet not choose sufficiently many vertices so that this will be good enough in terms of the approximation ratio achieved. Else, also because we know that this subgraph is large compared to $G$, if the second oracle returned a "large" solution $S_2$, then we know that every optimal solution must take many vertices from this subgraph, and hence, to compensate for this, the optimum of $G[S_1]$ must be "very small". So, we compute a 2-approximate solution $A$ to $G[S_1]$, which we know should not be "too large", and output the union of $A$ and $V(G) \setminus S_1$ (which yields a vertex cover).

## 3.3 Near-Linear Volume and Pure Lossy Kernelization Protocol for $d$-Hitting Set

For any fixed $\epsilon > 0$, we present a pure $d(1 - h(d, \epsilon))$-approximate (randomized) kernelization protocol for $d$-Hitting Set with call size $\mathcal{O}((\mathsf{frac})^{1+\epsilon})$ where $h(d, \epsilon)$ is a fixed positive constant that depends only on $d, \epsilon$. On a high-level, the idea of our more general lossy kernelization protocol is to compute a nested family of solutions based on the approach described above for Vertex Cover (see Fig. 2). Intuitively, as we now can sample only few sets (that is, $\mathsf{frac}^{1+\epsilon}$), when we compute a solution that hits them using an oracle call, the number of sets it misses can still be huge, and hence we will need to iteratively use the oracle (a constant number of times) until we reach a subuniverse such that we can optimally solve the subinstance induced by it by a single oracle call. Below, we give a more elaborate overview.

First, we apply our linear-element lossy kernel to have an instance $I_0 = (U_0, \mathcal{T}_0)$ where the universe $U_0$ consists of at most $d\mathsf{frac}$ elements. Here, the error of this application is not multiplied by the error attained next, but will only yield (as mentioned earlier) a negligible additive error (or directly a solution of the desired approximation ratio). The purpose of having only $d\mathsf{frac}$ elements is twofold, similarly as it is in the protocol described earlier for Vertex Cover. Afterwards, we begin by sampling a family $\mathcal{F}_1$ of roughly $\mathsf{frac}^{1+\epsilon}$ sets from $\mathcal{T}_0$. Then, we call the oracle on the sampled family $\mathcal{F}_1$ to obtain a solution $S_1$ to it. In case that solution $S_1$ is "large" (sufficiently larger than $|U_0|/d \leq \mathsf{frac}$), we can just return $U_0$. Else, we know that the family of sets corresponding to the subinstance $I_1$ induced by $U_1 = U_0 \setminus S_1$ – that is, the family of all sets in $\mathcal{T}_0$ contained in $U_1$, which we denote by $\mathcal{T}_1$ –

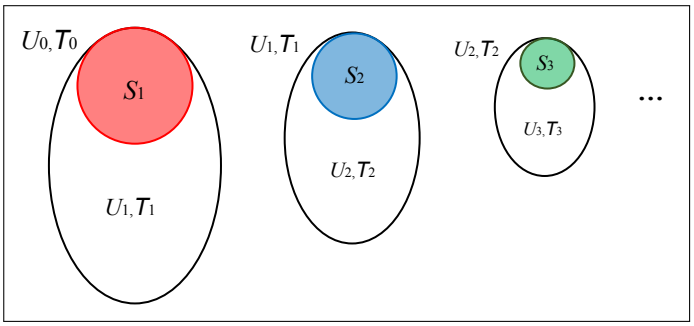



**Figure 2** The nested solutions computed by oracle calls in our lossy kernelization protocol for $d$-Hitting Set. Each $S_i$ is a solution to a subinstance $(U_{i-1}, \mathcal{F}_{i-1})$ sampled from $(U_{i-1}, \mathcal{T}_{i-1})$.

was missed by our set sample. In addition, we can show (due to the initial kernelization) that with high probability, every family of sets of size (roughly) at least $\mathsf{frac}^{d-\epsilon}$ that corresponds to a subinstance induced by a subset of $U_0$ has been hit by our set sample. Together, this implies that $\mathcal{T}_1$ has at most $\mathsf{frac}^{d-\epsilon}$ (rather than $\mathsf{frac}^{d}$) sets. Hence, in some sense, we have made progress towards the discovery of a sparse subinstance that we can optimally solve.

Due to important differences, let us describe also the second iteration – among at most $\frac{1}{\epsilon}(d-1)$ iterations performed in total – before skipping to the (last) one where we have a subinstance that we can optimally solve by an oracle call. The last iteration may not even be reached, if we find a "good enough" solution earlier. We remark that it is critical to stop and return a solution as soon as we find a "large enough" one by an oracle call[5] as for our arguments to work, we need to always deal with subinstances whose universe is large (a linear fraction of $|U_0|$), and these are attained by removing oracle solutions we got along the way. We begin the second iteration by sampling a family $\mathcal{F}_2$ of roughly $\mathsf{frac}^{1+\epsilon}$ sets from $\mathcal{T}_1$. Then, we call the oracle on the sampled family $\mathcal{F}_2$ to obtain a solution $S_2$ to it. On the one hand, in case that solution $S_2$ is "large" (sufficiently larger than $|U_1|/d$), we cannot just return $U_0$ as in the first iteration, as now it may not be true that the optimum of $I_0$ is large compared to $|U_0|$. Still, it is true that the optimum of $I_1$ is large compared to $|U_1|$. So, every optimal solution (to $I_0$) must take many elements from $U_1 \setminus S_2$, and hence, to compensate for this, the optimum of the subinstance induced by $S_1$ must be "very small". So, we compute a $d$-approximate solution to this subinstance, which we know should not be "too large" , and output the union of it and $U_1$ (which yields a hitting set). On the other hand, in case $S_2$ is "small", we proceed as follows. We observe that the family of sets corresponding to the subinstance $I_2$ induced by $U_2 = U_1 \setminus S_2$, whose family of sets we denote by $\mathcal{T}_2$, was missed by our set sample. In addition, we can show (due to the initial kernelization) that with high probability, every family of sets of size (roughly) at least $\mathsf{frac}^{d-2\epsilon}$ that corresponds to a subinstance induced by a subset of $U_1$ has been hit by our set sample. Together, this implies that $\mathcal{T}_2$ has at most $\mathsf{frac}^{d-2\epsilon}$ (rather than just $\mathsf{frac}^{d-\epsilon}$ as in the first iteration) sets. Hence, in some sense, we have made further progress towards the discovery of a sparse subinstnace that we can optimally solve.

Finally, we arrive at a subinstance $I'$ induced by a subuniverse $U' \subseteq U_0$ that is of size linear in $U_0$ (else we should have returned a solution earlier) and where the family of sets, $\mathcal{F}'$, is of size at most $\mathsf{frac}^{1+\epsilon}$. Then, we call the oracle on $I'$ to obtain a solution $S'$ to it. On

[5] The solution we return is not the one given by the oracle call, but its union with another solution, as will be clarified immediately, or just $U_0$ in case of the first iteration describe above.

the one hand, in case that solution $S'$ is "large" (sufficiently larger than $|U'|/d$), we compute a $d$-approximate solution to the subinstance induced by $U_0 \setminus U'$ (which is the union of all solutions returned by oracle calls except the last one), and output the union of it and $U'$. Otherwise, we output $(U_0 \setminus U') \cup S'$, which is "good enough" because $U'$ is sufficiently large while $S'$ is sufficiently small compared to it, it does not contain a "large enough" number of elements from $U_0$.

## 3.4 Outlook: Relation to Ruzsa-Szemerédi Graphs

A graph $G$ is an $(r,t)$-*Ruzsa-Szemerédi graph* if its edge set can be partitioned into $t$ edge-disjoint induced matchings, each of size $r$. These graphs were introduced in 1978 [28], and have been extensively studied since then. When $r$ is a function of $n$, let $\gamma(r)$ denote the maximum $t$ (which is a function of $n$) such that there exists an $(r,t)$-Ruzsa-Szemerédi graph. In [19], the authors considered the case where $r = cn$. They showed that when $c = \frac{1}{4}$, $\gamma(r) \in \Theta(\log n)$, and when $\frac{1}{5} \leq c \leq \frac{1}{4}$, $t \in \mathcal{O}(\frac{n}{\log n})$. It is an open problem whether whenever $c$ is a fixed constant, $t \in \mathcal{O}(n^{1-\epsilon})$. For any fixed constant $0 < c < \frac{1}{4}$, we present a $(1+4c)$-approximate (randomized) kernelization protocol for Vertex Cover with $t+1$ rounds and call size $\mathcal{O}(t(\mathsf{frac})^{1.5})$. Clearly, this result makes sense only when $t \in o(\sqrt{n})$, preferably $t \in \mathcal{O}(n^{\frac{1}{2}-\lambda})$ for $\lambda$ as close to $1/2$ as possible, because the volume is $\mathcal{O}(\mathsf{opt}^{2-\lambda})$. If $t$ is "sufficiently small" (depending on the desired number of rounds) whenever $c$ is a fixed constant (specifically, substitute $c = \frac{\epsilon}{4}$), this yields a $(1+\epsilon)$-approximate kernelization protocol.

We observe that, for a graph $G$, $r = r(n), t = t(n) \in \mathbb{N}$ and $U_1, U_2, \ldots, U_t \subseteq V(G)$ such that for all $i \in \{1,2,\ldots,t\}$, $G[U_i]$ has a matching $M_i$ of size at least $r$, and for all distinct $i,j \in \{1,2,\ldots,t\}$, $E(G[U_i]) \cap E(G[U_j]) = \emptyset$, we have that $G$ is a supergraph of an $(r,t)$-Ruzsa-Szemerédi graph. Having this observation in mind, we devise our protocol as follows. After applying an exact $2\mathsf{frac}$-vertex kernel, we initialize $E' = \emptyset$, and we perform $t+1$ iterations of the following procedure. We sample a set of roughly $\mathsf{frac}^{1.5}$ edges from $G$, and call the oracle on the subgraph of $G$ whose edge set is the set of samples edges union $E'$ to obtain a solution $S$ to it (but not to $G$), and compute a maximal matching $M$ in $G - S$. If $|M|$ is smaller than $cn \leq 2c\mathsf{frac}$, then we return the union of the set of vertices incident to edges in $M$ (which is a solution to $G - S$) and $S$. Else, similarly to the first protocol we described for Vertex Cover, we can show that with high probability, $G - S$ has (roughly) at most $k^{1.5}$ edges, and we add this set of edges to $E'$. The crux of the proof is in the argument that, at the latest, at the $(t+1)$-st iteration the computed matching will be of size smaller than $cn \leq 2c\mathsf{frac}$, as otherwise we can use the matchings we found, together with the vertex sets (of the form $G - S$) we found them in, to construct an $(r, t+1)$-Ruzsa-Szemerédi graph based on the aforementioned observation, which contradicts the choice of $t$.

## 3.5 $(1+\epsilon)$-Approximate $\mathcal{O}(\frac{1}{\epsilon} \cdot \mathsf{opt})$-Vertex Kernel for Implicit 3-Hitting Set Problems

Both of our lossy kernels share a common scheme, which might be useful to derive $(1+\epsilon)$-approximate linear-vertex kernels for other implicit hitting and packing problems as well. Essentially, they both consist of two rules (although in the presentation, they are merged for simplicity). To present them, we remind that a module (in a graph) is a set of vertices having the same neighborhood relations with all vertices outside the set. Now, our first rule reveals some modules in the graph, and our second rule shrinks their size. The first rule in both of our lossy kernels is essentially the same.

Now, we elaborate on the first rule. We start by computing an optimal solution $\alpha$ to the LP-relaxation of the corresponding 3-Hitting Set problem. Notice that $\mathsf{support}(\alpha)$ is a solution, and its size is at most $3\mathsf{frac}$ (in fact, we show that it is at most $3\mathsf{frac} - 2|\alpha^{-1}(1)|$). Then, the first rule is as follows. At the beginning, no vertex is marked. Afterwards, one-by-one, for each vertex $v$ assigned 1 by $\alpha$ (i.e., which belongs to $\alpha^{-1}(1)$), we construct a graph whose vertex set is the set of yet unmarked vertices in $V(G) \setminus \mathsf{support}(\alpha)$ and where there is an edge between every two vertices that create an obstruction together with $v$ (that is, an induced $P_3$ in Cluster Vertex Deletion and a triangle in Feedback Vertex Set in Tournaments). We compute a maximal matching in this graph, and decrease its size to $\frac{1}{\epsilon}$ if it is larger (in which case, it is no longer maximal). The vertices incident to the edges in the matching are then considered marked. We prove that among the vertices in $\alpha^{-1}(1)$ whose matching size was decreased, whose set is denoted by $D$, any solution can only exclude an $\epsilon$ fraction of its size among the vertices in $D$, and hence it is "safe" (in a lossy sense) to delete $D$. Let $M$ be the set of all marked vertices. Then, we show that $(\mathsf{support}(\alpha) \cup M) \setminus \{v\}$, for any $v \in \mathsf{support}(\alpha)$ (including those not in $\alpha^{-1}(1)$), is also a solution.

For Cluster Vertex Deletion, we prove that the outcome of the first rule means that the vertex set of every clique in $G - (\mathsf{support}(\alpha) \cup M)$ is a module in $G - D$, and that for every vertex in $\mathsf{support}(\alpha)$, the set of its neighbors in $V(G - (\mathsf{support}(\alpha) \cup M))$ is the vertex set of exactly one of these cliques. So, for Cluster Vertex Deletion, this gives rise to the following second reduction rule (which is, in fact, exact) to decrease the size of module. For every clique among the aforementioned cliques whose size is larger than that of its neighborhood, we arbitrarily remove some of its vertices so that its size will be equal to the size of its neighborhood. This rule is safe since if at least one of the vertices in such a clique is deleted by a solution, then because it is a module, either that deletion is irrelevant or the entire clique is deleted, and in the second case we might just as well delete its neighborhood instead. Because the neighborhoods of the cliques are pairwise-disjoint (since for every vertex in $\mathsf{support}(\alpha)$, the set of its neighbors in $V(G - (\mathsf{support}(\alpha) \cup M))$ is the vertex set of exactly one of the cliques), this means that now their total size is at most $(\mathsf{support}(\alpha) \setminus D) \cup M$, and hence we arrive at the desired kernel.

For Feedback Vertex Set in Tournaments, we consider the unique (because $G$ is a tournament) topological ordering of the vertices in $G - \mathsf{support}(\alpha)$, so that all arcs are "forward" arcs. We prove that the outcome of the first rule means that each vertex $v \in \mathsf{support}(\alpha)$ has a unique position within this ordering when restricted to $G - (\mathsf{support}(\alpha) \cup M)$, so that still all arcs (that is, including those incident to $v$) are forward arcs in $G - (\mathsf{support}(\alpha) \cup M) \cup \{v\}$. (Further, the vertex set of each subtournament induced by the vertices "between" any two marked vertices in $G - \mathsf{support}(\alpha)$ is a module in $G - D$.) We are thus able to characterize all triangles in $G - D$ as follows: each either consists of three vertices in $(\mathsf{support}(\alpha) \setminus D) \cup M$, or it consists of a vertex $v \in \mathsf{support}(\alpha) \setminus D$, a vertex $u \in (\mathsf{support}(\alpha) \setminus D) \cup M$ and a vertex $w \in V(G) \setminus (\mathsf{support}(\alpha) \cup M)$ with a backward arc between $v$ and $u$ and where $w$ is "in-between" the positions of $v$ and $u$. This gives rise to a reduction rule for module shrinkage whose presentation and analysis are more technical than that of Cluster Vertex Deletion (in particular, unlike the second rule of Cluster Vertex Deletion, the second rule of Feedback Vertex Set in Tournaments is lossy) and of the first rule; hence, due to lack of space, we omit them.

## 4 Conclusion

In this paper, we presented positive results on the kernelization complexity of $d$-Hitting Set, as well as for its special cases Cluster Vertex Deletion and Feedback Vertex Set in Tournaments. First, we proved that if we allow the kernelization to be *lossy* with

a qualitatively better loss than the best possible approximation ratio of polynomial time approximation algorithms, then one can obtain kernels where the number of elements is linear for every fixed $d$. Further, we extended the notion of lossy kernelization algorithms to *lossy kernelization protocols* and, then, presented our main result: For any $\epsilon > 0$, $d$-Hitting Set admits a (randomized) pure $(d - \delta)$-approximate kernelization protocol of call size $\mathcal{O}(k^{1+\epsilon})$. Here, the number of rounds and $\delta$ are fixed constants (that depend only on $d$ and $\epsilon$). Finally, we complemented the aforementioned results as follows: for the special cases of 3-Hitting Set, namely, Cluster Vertex Deletion and Feedback Vertex Set in Tournaments, we showed that for any $0 < \epsilon < 1$, they admits a $(1+\epsilon)$-approximate $\mathcal{O}(\frac{1}{\epsilon} \cdot \mathsf{opt})$-vertex kernel.

We conclude the paper with a few interesting open problems.

1. Does $d$-Hitting Set admit a kernel with $f(d) \cdot k^{d-1-\epsilon}$ elements for some fixed $\epsilon > 0$, or, even, with just $f(d) \cdot k$ elements?
2. Does $d$-Hitting Set admit a $(1+\epsilon)$-approximate $\mathcal{O}(f(\epsilon) \cdot k)$-element kernel (or protocol)?
3. Does $d$-Hitting Set admit a $(1+\epsilon)$-approximate $\mathcal{O}(f(\epsilon) \cdot k)$-bits kernel (or protocol)?
4. Do Feedback Vertex Set in Tournaments and Cluster Vertex Deletion admit linear vertex kernels?
5. Are lossy kernelization protocols "more powerful" than lossy kernelization algorithms?

## References


1. Faisal N. Abu-Khzam. A kernelization algorithm for $d$-Hitting Set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010. doi:10.1016/j.jcss.2009.09.002.
2. Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for feedback arc set in tournaments. *J. Comput. Syst. Sci.*, 77(6):1071–1078, 2011.
3. Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.
4. Hans L. Bodlaender, Fedor V. Fomin, and Saket Saurabh. Open problems, worker 2010. *Available at http://fpt.wikidot.com/open-problems*, 2010.
5. Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007, 2000.
6. Jianer Chen, Iyad A Kanj, and Weijia Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001.
7. Marek Cygan, Fedor V. Fomin, Bart MP Jansen, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Open problems for fpt school 2014. *URL: http://fptschool. mimuw. edu. pl/opl. pdf*, 2014.
8. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
9. Holger Dell and Dániel Marx. Kernelization of packing problems. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 68–81, 2012.
10. Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014.
11. Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.
12. Andrew Drucker. New limits to classical and quantum instance compression. *SIAM Journal on Computing*, 44(5):1443–1479, 2015.
13. Michael R. Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Ulrike Stege, Dimitrios M. Thilikos, and Sue Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. *Algorithmica*, 52(2):167–176, 2008.

14 Samuel Fiorini, Gwenaël Joret, and Oliver Schaudt. Improved approximation algorithms for hitting 3-vertex paths. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings*, volume 9682 of *Lecture Notes in Computer Science*, pages 238–249, 2016.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

16 Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-Hitting Set and 3-Set Packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019. doi:10.1145/3293466.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization. Theory of parameterized preprocessing*. Cambridge University Press, Cambridge, 2019.

18 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

19 Jacob Fox, Hao Huang, and Benny Sudakov. On graphs decomposable into induced matchings of linear sizes. *Bulletin of the London Mathematical Society*, 49(1):45–57, 2017.

20 Zoltán Füredi. Matchings and covers in hypergraphs. *Graphs and Combinatorics*, 4(1):115–206, 1988.

21 Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 104–113, 2012.

22 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.

23 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459, 2012. doi:10.1109/FOCS.2012.46.

24 Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1010–1018, 2020.

25 Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237, 2017.

26 Matthias Mnich, Virginia Vassilevska Williams, and László A. Végh. A 7/3-approximation for feedback vertex sets in tournaments. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 67:1–67:14, 2016.

27 George L Nemhauser and Leslie Earl Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical programming*, 6(1):48–61, 1974.

28 Imre Z Ruzsa and Endre Szemerédi. Triple systems with no six points carrying three triangles. *Combinatorics (Keszthely, 1976), Coll. Math. Soc. J. Bolyai*, 18:939–945, 1978.

29 Jie You, Jianxin Wang, and Yixin Cao. Approximate association via dissociation. *Discrete Applied Mathematics*, 219:202–209, 2017.

# Bootstrapping Dynamic Distance Oracles

**Sebastian Forster**
Department of Computer Science, University of Salzburg, Austria

**Gramoz Goranci**
Faculty of Computer Science, University of Vienna, Austria

**Yasamin Nazari**[1]
Department of Computer Science, VU Amsterdam, The Netherlands

**Antonis Skarlatos**
Department of Computer Science, University of Salzburg, Austria

## Abstract

Designing approximate all-pairs distance oracles in the fully dynamic setting is one of the central problems in dynamic graph algorithms. Despite extensive research on this topic, the first result breaking the $O(\sqrt{n})$ barrier on the update time for any non-trivial approximation was introduced only recently by Forster, Goranci and Henzinger [SODA'21] who achieved $m^{1/\rho+o(1)}$ amortized update time with a $O(\log n)^{3\rho-2}$ factor in the approximation ratio, for any parameter $\rho \geq 1$.

In this paper, we give the first *constant-stretch* fully dynamic distance oracle with small polynomial update and query time. Prior work required either at least a poly-logarithmic approximation or much larger update time. Our result gives a more fine-grained trade-off between stretch and update time, for instance we can achieve constant stretch of $O(\frac{1}{\rho^2})^{4/\rho}$ in amortized update time $\tilde{O}(n^{\rho})$, and query time $\tilde{O}(n^{\rho/8})$ for any constant parameter $0 < \rho < 1$. Our algorithm is randomized and assumes an oblivious adversary.

A core technical idea underlying our construction is to design a black-box reduction from decremental approximate hub-labeling schemes to fully dynamic distance oracles, which may be of independent interest. We then apply this reduction repeatedly to an existing decremental algorithm to bootstrap our fully dynamic solution.

**2012 ACM Subject Classification** Theory of computation → Dynamic graph algorithms

**Keywords and phrases** Dynamic graph algorithms, Distance Oracles, Shortest Paths

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.50

**Related Version** *Full Version*: https://arxiv.org/abs/2303.06102

**Funding** This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702).

## 1 Introduction

The All-Pairs Shortest Paths (APSP) problem is one of the cornerstone graph problems in combinatorial optimization. It has a wide range of applications, for instance in route planning, navigation systems, and routing in networks, and it has been extensively studied from both practical and theoretical perspectives. In theoretical computer science, this problem enjoys much popularity due to its historic contributions to the development of fundamental algorithmic tools and definitions as well as being used as a subroutine for solving other problems.

[1] This work was conducted when this author was a postdoc at University of Salzburg.





The APSP problem has also been studied extensively in *dynamic* settings. Here, the underlying graph undergoes edge insertions and deletions (referred to as edge *updates*), and the goal is to quickly report an approximation to the shortest paths between *any* source-target vertex pair. The dynamic setting is perhaps even more realistic for some of the applications of the APSP problem, e.g., in navigation systems, as link statistics of road networks are prone to changes because of evolving traffic conditions. A naive (but rather expensive) solution to handle these updates is achieved by running an exact static algorithm after each update. However, at an intuitive level, one would expect to somehow exploit the fact that a single update is small compared to the size of the network, and thus come up with much faster update times.

Much of the research literature in dynamic APSP has focused on the *partially* dynamic setting. In contrast to the *fully* dynamic counterpart, this weaker model restricts the types of updates to edge insertions or deletions only. Some reasons for studying partially dynamic algorithms include their application as a subroutine in speeding up static algorithms (e.g., flow problems [36]), or their utilization as a stepping stone for designing fully-dynamic algorithms, something that we will also exploit in this work. The popularity of the partially dynamic setting can also attributed to the fact that dealing with only one type of update usually leads to better algorithmic guarantees. In fact, the fully dynamic APSP problem admits strong conditional lower bounds in the *low approximation* regimes: under plausible hardness assumptions, Abboud and Vassilevska Williams [3], and later Henzinger, Krinninger, Nanongkai, and Saranurak [32] show that there are no dynamic APSP algorithms achieving a $(3-\epsilon)$ approximation with sublinear query time and the update time being a small polynomial.

From an upper bounds perspective, there are only two works that achieve sublinear update time for fully dynamic APSP. Abraham, Chechik, and Talwar [5] showed that there is an algorithm that achieves constant approximation and sublinear update time. However, their algorithm cannot break the $O(\sqrt{n})$ barrier on the update time. Forster, Goranci, and Henzinger [25] gave different trade-offs between approximation and update time. In particular, in $n^{o(1)}$ amortized update time and polylogarithmic query time they achieve $n^{o(1)}$ approximation. These two works suffer from either a large approximation guarantee or update time, leaving open the following key question:

*Is there a fully dynamic APSP algorithm that achieves* constant *approximation with a very small polynomial update and query time?*

## 1.1 Our result

In this paper, we answer the question of achieving constant approximation with a very small polynomial update time for the fully dynamic APSP in the affirmative, also known as the *fully dynamic distance oracle* problem. More generally, we obtain a trade-off between approximation, update time, and query time as follows:

▶ **Theorem 1.** *Given a weighted undirected graph $G = (V, E, w)$ with polynomial weights*[2]*, and a constant parameter $0 < \rho < 1$, there is a randomized fully dynamic distance oracle with constant stretch $(\frac{256}{\rho^2})^{4/\rho}$ that w.h.p. achieves $\tilde{O}(n^{\rho})$ amortized update time and $\tilde{O}(n^{\rho/8})$ query time. These guarantees hold against an oblivious adversary.*

[2] In this paper, we assume for ease of notation that the edge weights are integers in the range from 1 to $W$, where $W$ is polynomial in $n$. Using a standard approach (see e.g., [8]) this extends to rational edge weights in some range from the minimum weight $W_{\min}$ to the maximum weight $W_{\max}$, where $W_{\max}/W_{\min}$ is polynomial in $n$.

Our distance oracle can also be extended to report the actual (approximate) shortest path when answering queries (see the full version [26] for a sketch). In addition to the constant stretch regime, we obtain several interesting tradeoffs, as shown in Theorem 5. For example, our algorithm achieves $O(\log \log n)$ stretch with a much faster query time of $n^{o(1)}$ and very small polynomial update time (see Corollary 6).

Our result brings the algorithmic guarantees on fully dynamic distance oracles closer to the recent conditional hardness result by Abboud, Bringmann, Khoury, and Zamir [2] (and the subsequent refinement in [1]), who showed that there is no fully dynamic algorithm that simultaneously achieves constant approximation and $n^{o(1)}$ update and query time. We also remark that our results are consistent with their lower bound since if we insist on constant approximation, the above trade-off shows that the update time cannot be made as efficient as $n^{o(1)}$.

On the technical side, our result follows the widespread "high-level" approach of extending decremental algorithms to the fully dynamic setting (see e.g. [33, 38, 39, 40, 10, 30, 5, 25]) and it is inspired by recent developments on the dynamic distance oracle literature that rely on vertex sparsification [25, 18, 27]. Specifically, we design a reduction that turns a decremental hub-labeling scheme with some specific properties into a fully dynamic distance oracle, which may be of independent interest. Our key observation is that an existing state-of-the-art decremental distance oracle that works against an oblivious adversary can serve as such hub-labeling scheme. The fully dynamic distance oracle is then obtained by repeatedly applying the reduction whilst carefully tuning various parameters across levels in the hierarchy.

More generally, our reduction does not make any assumptions on the adversary and is based on properties that are quite natural. At a high-level, we consider decremental approximate hub-labeling schemes with the following properties. (1) For every vertex $v \in V$, maintain a set $S(v)$, called a *hub set*, that has bounded size. (2) For every vertex $v \in V$, maintain distance estimates $\delta(v, u)$ for each $u \in S(v)$, with bounded *recourse*, which is defined as the number of times such distance estimates are affected during the execution of the algorithm. (3) Return the final estimate between a pair of vertices $s, t \in V$, by minimizing estimates over elements in $S(s) \cap S(t)$.

Many known distance oracles (e.g. variants of the well-known distance oracle of [44]) have a query mechanism that satisfies the first and third properties, while efficient dynamic distance oracles are often based on bounded recourse structures satisfying the second property.

Hence we hope that this reduction can be further utilized in the future by characterizing deterministic decremental distance oracles or the ones with different stretch/time tradeoffs as such hub-labeling schemes. Similar reductions have been previously proposed in [5] and then refined in [25] in slightly different contexts. In this work, in addition to refining this approach for obtaining a constant stretch distance oracle, we aim to keep the reduction as modular as possible to facilitate potential future applications.

## 1.2 Related Work

In the following, we give an overview of existing works on fully dynamic all-pairs distance oracles by dividing them into several categories based on their stretch guarantee. Unless noted otherwise, all algorithms cited in the following are randomized and have amortized update time. We report running time bounds for constant accuracy parameter $\epsilon$ and assume that we are dealing with graphs with positive integer edge weights that are polynomial in the number of vertices. We would also like to point out that all "combinatorial" algorithms discussed in the following (i.e., algorithms that do not rely on "algebraic" techniques like

dynamic matrix inverse) are internally employing decremental algorithms. Decremental algorithms have also been studied on their own with various tradeoffs [40, 10, 31, 16, 35, 24], and competitive deterministic algorithms have been devised, e.g., [30, 11, 19].

**Exact.** After earlier attempts on the problem [34, 23], Demetrescu and Italiano [22] presented their seminal work on exact distance maintenance achieving $\tilde{O}(n^2)$ update time (with log-factor improvements by Thorup [42]) and constant query time for weighted directed graphs.

Subsequently, researchers have developed algorithms with subcubic worst-case update time and constant query time [43, 4] with some of them being deterministic [28, 17]. Note that one can construct a simple update sequence for which any fully dynamic algorithm maintaining the distance matrix or the shortest path matrix explicitly needs to perform $\Omega(n^2)$ changes to this matrix per update.

Algorithms breaking the $n^2$ barrier at the cost of large query time have been obtained in unweighted directed graphs by Roditty and Zwick [39] (update time $\tilde{O}(mn^2/t^2)$ and query time $O(t)$ for any $\sqrt{n} \leq t \leq n^{3/4}$), Sankowski [41] (worst-case update time $O(n^{1.897})$ and query time $O(n^{1.265})$), and van den Brand, Nanongkai, and Saranurak [15] (worst-case update time $O(n^{1.724})$ and query time $O(n^{1.724})$). The latter two approaches are algebraic and their running time bounds depend on the matrix multiplication coefficient $\omega$.

**$(1+\epsilon)$-approximation.** In addition to exact algorithms, combinatorial and algebraic algorithms have also been developed for the low stretch regime of $(1+\epsilon)$-approximation. In particular, Roditty and Zwick [40] obtained the following trade-off with a combinatorial algorithm: update time $\tilde{O}(mn/t)$ and query time $O(t)$ for any $\delta > 0$ and $t \leq m^{1/2-\delta}$. Subsequently, for $t \leq \sqrt{n}$, a deterministic variant was developed [30] and it was generalized to weighted directed graphs [10]. Furthermore, by a standard reduction (see e.g. [12]) using a decremental approximate single-source shortest paths algorithm [31, 11], one obtains a combinatorial, deterministic algorithm with update time $O(nm^{1+o(1)}/t)$ and query time $O(t)$ for any $t \leq n$, for the fully dynamic all-pairs problem in weighted undirected graphs. Conditional lower bounds [37, 3, 32] suggest that the update and the query time cannot be both small polynomials in $n$. For example, no algorithm can maintain a $(5/3-\epsilon)$-approximation with update time $O(m^{1/2-\delta})$ and query time $O(m^{1-\delta})$ for any $\delta > 0$, unless the OMv conjecture fails [32].

Algebraic approaches can achieve subquadratic update time and sublinear query time, namely worst-case update time $O(n^{1.863})$ and query time $O(n^{0.666})$ in weighted directed graphs [14], or worst-case update time $O(n^{1.788})$ and query time $O(n^{0.45})$ in unweighted undirected graphs [13]. As the conditional lower bound by Abboud and Vassilevska Williams [3] shows, algebraic approaches seem to be necessary in this regime: unless one is able to multiply two $n \times n$ Boolean matrices in $O(n^{3-\delta})$ time for some constant $\delta > 0$, no fully dynamic algorithm for *st* reachability in directed graphs can have $O(n^{2-\delta'})$ update and query time and $O(n^{3-\delta'})$ preprocessing time (for some constant $\delta' > 0$). While not explicitly stated in the paper, the same conditional lower bound extends to fully dynamic $(1+\epsilon)$-approximate *st* distances on undirected unweighted graphs for a small enough constant $\epsilon$.

**$(2+\epsilon)$-approximation.** Apart from earlier work [34], the only relevant algorithm in the $(2+\epsilon)$-approximation regime is by Bernstein [9] and achieves update time $m^{1+o(1)}$ and query time $O(\log\log\log n)$ in weighted undirected graphs. It can be made deterministic using the deterministic approximate single-source shortest path algorithm by Bernstein, Probst Gutenberg, and Saranurak [11]. The only conditional lower bound in this regime that we are aware of states that no algorithm can maintain a $(3-\epsilon)$-approximation with update time $O(n^{1/2-\delta})$ and query time $O(n^{1-\delta})$ for any $\delta > 0$, unless the OMv conjecture fails [32].

**Larger approximation.** In the regime of stretch at least 3, the following trade-offs between stretch and update time have been developed: Abraham, Chechik, and Talwar [5] designed an algorithm for unweighted undirected graphs with stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2}n^{1/k})$, and query time $O(k^2\rho^2)$, where $k \geq 1$ is a freely chosen parameter and $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$. Forster, Goranci, and Henzinger [25] designed an algorithm for weighted undirected graphs with stretch $O(\log n)^{3k-2}$, update time $m^{1/k+o(1)} \cdot O(\log n)^{4k-2}$, and query time $O(k(\log n)^2)$, where $k \geq 2$ is an arbitrary integer parameter. Very recently, Chuzhoy and Zhang [20] independently obtained a deterministic algorithm for weighted undirected graphs with stretch $(\log\log n)^{2^{1/\rho^3}}$, update time $\tilde{O}(n^{O(\rho)})$, and query time $\tilde{O}(2^{O(1/\rho)})$ for any choice of $\frac{2}{(\log n)^{1/200}} < \rho < \frac{1}{400}$. Similar to our work, they also achieve sublogarithmic stretch but their guarantee cannot be reduced all the way to a constant. While our algorithm has the advantage of achieving constant stretch, their algorithm is deterministic, and thus works against an adaptive adversary. Finally, note that any algorithm whose update time depends on the sparsity of the graph (possibly also a static one) can be run on a spanner of the input graph maintained by a fully dynamic spanner algorithm [7]. These upper bounds are complemented by the following conditional lower bound: for any integer constant $k \geq 2$, there is no dynamic approximate distance oracle with stretch $2k-1$, update time $O(m^u)$ and query time $O(m^q)$ with $ku + (k+1)q < 1$, unless the 3-SUM conjecture fails [1].

## 2 Preliminaries

We consider weighted undirected graphs $G = (V, E, w)$ with positive integer edge weights. We denote by $n = |V|$ the number of vertices, by $m = |E|$ the number of edges, and by $W$ the maximum weight of an edge. For every pair of vertices $u, v \in V$, the *distance* $\text{dist}_G(u, v)$ between $u$ and $v$ in $G$ is the length of a shortest path from $u$ to $v$ in $G$. For a path $P$, we denote by $w_G(P)$ the length of $P$ in $G$, by $E(P)$ the edges of $P$, and by $|P| = |E(P)|$ the number of edges of $P$. Also for a graph $H$, we denote by $V(H)$ and $E(H)$ the vertex and the edge set of $H$ respectively.

In dynamic graph algorithms, the graph is subject to updates and the algorithm has to process these updates by spending as little time as possible. In this paper, we consider updates that insert a single edge to the graph or delete a single edge from the graph. Moreover, observe that an update that changes the weight of an edge can be simulated by two updates, where the first update deletes the corresponding edge and the second update re-inserts the edge with the new weight. Let $G^{(0)}$ be the initial graph, and $G^{(\tau)}$ be the graph at time $\tau$ which is the time after $\tau$ updates have been performed to the graph.

In this paper we are interested in designing *fully dynamic* algorithms which can process edge insertions and edge deletions, and thus, weight changes as well. A *decremental* algorithm can process only edge deletions and weight increases. We assume that the updates to the graph are performed by an *oblivious adversary* who fixes the sequence of updates before the algorithm starts. Namely, the adversary cannot adapt the updates based on the choices of the algorithm during the execution. We say that an algorithm has *amortized update time* $u(n, m)$ if its total time spent for processing any sequence of $\ell$ updates is bounded by $\ell \cdot u(n, m)$, when it starts from an empty graph with $n$ vertices and during all the updates has at most $m$ edges (the time needed to initialize the algorithm on the empty graph before the first update is also included). An algorithm is *path reporting* if after a query can also return the corresponding path explicitly.

In our analysis we use $\tilde{O}(1)$ to hide factors polylogarithmic in $nW$. Namely, we write $\tilde{O}(1)^d$ to represent the term $O(\log^{cd} nW)$, for a constant $c$ and a parameter $d$.

# 3 Fully Dynamic Distance Oracle

The technical details of our distance oracle are divided into three parts. Initially in Section 3.1, we give the definition of a hub-labeling scheme together with other useful definitions. Afterwards, we provide a reduction for extending a decremental approximate hub-labeling scheme with some properties to a fully dynamic distance oracle. Then in Section 3.2, we explain how an existing decremental algorithm gives us an approximate hub-labeling scheme that we can use in this reduction, and finally in Section 3.3 we put everything together by applying our reduction repeatedly, in order to get a family of fully dynamic distance oracles.

## 3.1 From decremental hub-labeling scheme to fully dynamic distance oracle via reduction

We start by defining approximate hub-labeling schemes, and then explain how they are used in our reduction. Hub-labeling schemes were formally defined by [6] (and were previously introduced under the name 2-hop cover[3] in [21]). We are slightly modifying the definition for our purpose, for instance by considering an approximate variant.

▶ **Definition 2** (Approximate Hub-Labeling Scheme). *Given a graph $G = (V, E)$, a* hub-labeling scheme $\mathcal{L}$ *of* stretch $\alpha$ *consists of*

1. *for every vertex $v \in V$, a* hub set $S(v) \subseteq V$ *and*
2. *for every pair of vertices $u, v \in V$, a distance estimate $\delta(v, u)$ such that $\text{dist}_G(v, u) \leq \delta(v, u) < \infty$ if $u \in S(v)$ and $\delta(v, u) = \infty$ otherwise.*

*and for every pair of vertices $s$ and $t$ guarantees that*

$$\delta_{\mathcal{L}}(s, t) := \min_{v \in S(s) \cap S(t)} (\delta(s, v) + \delta(t, v)) \leq \alpha \cdot \text{dist}_G(s, t)\,.$$

The *distance label* of a vertex $v$ consists of the hub set $S(v)$ and the corresponding distance estimates $\delta(v, u)$, for all $u \in S(v)$.

Note that the definition implies $\delta_{\mathcal{L}}(s, t) \geq \text{dist}_G(s, t)$ for every pair of vertices $s$ and $t$. Furthermore, a hub-labeling scheme of stretch $\alpha$ directly implements a distance oracle of stretch $\alpha$ with query time $O(\max_{v \in V} |S(v)|)$ that consists of the collection of distance labels for all vertices $v \in V$. We also remark that the entries of value $\infty$ in the distance estimate $\delta(\cdot, \cdot)$ do not need to be stored explicitly if the hub sets are stored explicitly and that the distance estimate $\delta(\cdot, \cdot)$ is not necessarily symmetric.

In the following we consider *decremental* algorithms for maintaining approximate hub-labeling schemes, that is, *decremental approximate hub-labeling schemes* which process each edge deletion in the graph by first updating their internal data structures and then outputting the changes made to the hub sets and the distance estimates $\delta(\cdot, \cdot)$. Namely for a vertex $v \in V$, vertices may leave or join $S(v)$, or the distance estimates of vertices belonging to $S(v)$ may change, since the decremental algorithm has to update this information for maintaining correctness at query time.

Denote by $S^{(\tau)}(v)$ the hub set of a vertex $v \in V$ after $\tau$ updates have been processed by the decremental approximate hub-labeling scheme (we may omit the superscript $\tau$ whenever time is fixed), where $\tau \geq 1$ is an integer parameter. Then for a pair of vertices $u, v \in V$, the distance estimate $\delta(v, u)$ after $\tau$ updates is defined based on Definition 2 and $S^{(\tau)}(v)$. Namely, if $u$ is inside the hub set of $v$ after $\tau$ updates (i.e., $u \in S^{(\tau)}(v)$) then $\text{dist}_{G^{(\tau)}}(v, u) \leq \delta(v, u) < \infty$, otherwise $\delta(v, u) = \infty$.

[3] The concept of 2-hop cover or hub-labeling should not be confused with the (related) concept of a hopset that we will later see in Section 3.2.

After $\tau$ edge deletions have been processed by the decremental approximate hub-labeling scheme, there are three possible types of changes to the distance estimates $\delta(v,\cdot)$ that correspond to a vertex $v \in V$, due to the last edge deletion. (1) The distance estimate $\delta(v,u)$ changes for a vertex $u \in S^{(\tau-1)}(v) \cap S^{(\tau)}(v)$ that remains inside the hub set of $v$. (2) The distance estimate $\delta(v,u)$ becomes $\infty$ because a vertex $u \in S^{(\tau-1)}(v) \setminus S^{(\tau)}(v)$ leaves the hub set of $v$. (3) The distance estimate $\delta(v,u)$ receives a finite value because a vertex $u \in S^{(\tau)}(v) \setminus S^{(\tau-1)}(v)$ enters the hub set of $v$. Let $\chi^{(\tau)}(v)$ be the number of all these changes to $\delta(v,\cdot)$ corresponding to $v$ at time $\tau$. In other words, for a fixed vertex $v \in V$, the value of $\chi^{(\tau)}(v)$ is equal to the number of vertices $u$ whose corresponding value of $\delta(v,u)$ changes due to the last edge deletion. Moreover, let $X(v) = \sum_\tau \chi^{(\tau)}(v)$ be the total number of such changes to $\delta(v,\cdot)$ corresponding to $v$ over the course of the algorithm.

In the following lemma, we present a reduction from a decremental approximate hub-labeling scheme to a fully dynamic distance oracle.

▶ **Lemma 3.** *Consider a decremental hub-labeling scheme $\mathcal{A}$ of stretch $\alpha$ with total update time $T_{\mathcal{A}}(n,m,W)$ and query time $Q_{\mathcal{A}}(n,m,W)$, with the following properties:*

1. *$\forall v \in V$ and $\forall \tau : |S^{(\tau)}(v)| \leq \gamma$. In other words, the size of the hub set of any vertex is bounded by $\gamma$ at any moment of the algorithm.*
2. *$\forall v \in V : X(v) \leq \zeta$. In other words, for every vertex $v \in V$ the total number of changes to $\delta(v,\cdot)$ is at most $\zeta$ over the course of the algorithm. Moreover the algorithm detects and reports these changes explicitly.*

*Then given $\mathcal{A}$ and a fully dynamic distance oracle $\mathcal{B}$ of stretch $\beta$ with amortized update time $t_{\mathcal{B}}(n,m,W)$ and query time $Q_{\mathcal{B}}(n,m,W)$, for any integer $\ell \geq 1$, there is a fully dynamic distance oracle $\mathcal{C}$ of stretch $\alpha\beta$ with amortized update time $t_{\mathcal{C}}(n,m,W) = T_{\mathcal{A}}(n,m,W)/\ell + t_{\mathcal{B}}(\min(\ell(2+2\mu),n), \ell(1+2\mu), nW) \cdot (2+4\mu)$ and query time $Q_{\mathcal{C}}(n,m,W) = Q_{\mathcal{A}}(n,m,W) + \gamma^2 \cdot Q_{\mathcal{B}}(\min(\ell(2+2\mu),n), \ell(1+2\mu), nW)$, where $\mu = \gamma + \zeta$.*

**Proof.** We organize the proof in three parts. The first part gives the reduction from $\mathcal{A}$ and $\mathcal{B}$ to $\mathcal{C}$, and the second and third part concern the correctness and the running times respectively.

**Reduction.** The fully dynamic distance oracle $\mathcal{C}$ proceeds in phases of length $\ell$. For each phase, we denote by $\tau$ the number of updates processed by $\mathcal{A}$ during the phase. At the beginning of the first phase (which is also the beginning of the algorithm), $\mathcal{C}$ initializes the fully dynamic distance oracle $\mathcal{B}$ on the initially empty graph $G$ consisting of $2\ell$ vertices[4], and sets an update counter to 0. Whenever an update to $G$ occurs in the first phase, the update is directly processed by $\mathcal{B}$.[5] As soon as the number of updates is more than $\ell$, the second phase is started. We define several sets and the graph $H$ that the fully dynamic distance oracle $\mathcal{C}$ maintains during each subsequent phase:

- Let $F$ be the set of edges present in $G$ at the beginning of the phase, $E$ be the current set of edges in $G$, and $D$ be the set of edges deleted from $G$ during the phase.
- Let $I = E \setminus (F \setminus D)$ be the set of edges inserted to $G$ since the beginning of the phase without subsequently having been deleted during the phase, and $U = \{v \in V \mid \exists e \in I : v \in e\}$ be the set of endpoints of edges in $I$.

[4] This minor technical detail makes sure that $\mathcal{B}$ does not have to deal with vertex insertions.
[5] The special treatment of the first $\ell$ updates is just a technical necessity for a rigorous amortization argument in the running time analysis.

- Let $H$ be the auxiliary graph that consists of all edges $(u, v) \in I$, together with their hub sets $S^{(\tau)}(u)$ and $S^{(\tau)}(v)$ after $\tau$ edge deletions have been processed by $\mathcal{A}$. Specifically, $V(H) = \{v \in V \mid v \in U \text{ or } (u \in U \text{ and } v \in S^{(\tau)}(u))\}$ and $E(H) = \{(u, v) \mid (u, v) \in I \text{ or } (v \in U \text{ and } u \in S^{(\tau)}(v))\}$. Note that at any fixed moment, the size of $V(H)$ is at most $\ell \cdot (2 + 2\gamma)$ and the size of $E(H)$ is at most $\ell \cdot (1 + 2\gamma)$.

At the beginning of each subsequent phase, $\mathcal{C}$ stores the sets $F, E, D, I, U$, and the auxiliary graph $H$, and sets an update counter to 0. Furthermore, $\mathcal{C}$ initializes the decremental approximate hub-labeling scheme $\mathcal{A}$ on the current graph $G$, and the fully dynamic distance oracle $\mathcal{B}$ on $H$ which is initially an empty "sketch" graph on $\ell \cdot (2 + 2\mu)$ vertices. The graph $H$ can be thought of as responsible for maintaining estimates for paths that use inserted edges.

Whenever an update to $G$ occurs, $\mathcal{C}$ first checks via the update counter whether the number of updates since the beginning of the phase is more than $\ell$. If this is the case, then $\mathcal{C}$ starts a new phase. Otherwise, after an update the fully dynamic distance oracle $\mathcal{C}$ does the following. On the insertion of an edge $(u, v)$ to $G$, $\mathcal{C}$ adds $(u, v)$ to $I$, adds $u$ and $v$ to $U$, and adds the edge $(u, v)$ to $H$, together with the edges $(u, p)$ for every $p \in S^{(\tau)}(u)$ and $(v, p)$ for every $p \in S^{(\tau)}(v)$. Any time an edge $(u, v)$ is added to $H$, its weight is set to:

$$w_H(u, v) = \min(w_G(u, v), \delta(u, v), \delta(v, u)).$$

Whenever the first edge incident to some vertex $v$ is added to $H$, the algorithm finds a "fresh" vertex (of degree 0) in $H$ and henceforth identifies it as $v$. This is always possible, since by the two properties of $\mathcal{A}$, the number of such vertices in a phase of length $\ell$ is at most $\ell \cdot (2 + 2\mu)$.

On the deletion of an edge $(u, v) \in E$ from $G$, there are two cases to consider.

1. If the edge $(u, v)$ was not present at the beginning of the current phase, or has been deleted and re-inserted (i.e., $(u, v) \in I$), then $\mathcal{C}$ removes $(u, v)$ from $I$, adds $(u, v)$ to $D$, and updates the set $U$ and the graph $H$ accordingly. In particular, if $u \in U$ and $v \in S^{(\tau)}(u)$, or $v \in U$ and $u \in S^{(\tau)}(v)$, $\mathcal{C}$ updates the weight of the edge $(u, v)$ in $H$ to $w_H(u, v) = \min(\delta(u, v), \delta(v, u))$ (as $w_G(u, v) = \infty$ after the deletion), otherwise $\mathcal{C}$ removes $(u, v)$ from $H$. Also, for all the vertices $v$ that left $U$ and all the edges $(v, p) \in E(H)$ such that $p \in S^{(\tau)}(v)$, if $p \in U$ and $v \in S^{(\tau)}(p)$, then $\mathcal{C}$ updates the weight of $(v, p)$ in $H$ to $w_H(v, p) = \delta(p, v)$ (as $v \notin U$ after the deletion), and otherwise $\mathcal{C}$ removes $(v, p)$ from $H$.
2. If the edge $(u, v)$ was present at the beginning of the current phase and has not been deleted yet (i.e., $(u, v) \in F \setminus D$), then $\mathcal{C}$ adds $(u, v)$ to $D$ and the deletion is processed by $\mathcal{A}$. Whenever $\mathcal{A}$ changes some distance estimates $\delta(v, \cdot)$ that correspond to a vertex $v \in U$ (i.e, $v$ is a vertex of $H$ and an endpoint of an edge in $I$) and its hub set, $\mathcal{C}$ updates the graph $H$ accordingly. In particular, there are three possible scenarios at time $\tau$ of $\mathcal{A}$.[6] (1) Whenever the value of $\delta(v, u)$ changes for a vertex $u \in S^{(\tau-1)}(v) \cap S^{(\tau)}(v)$ that remains inside the hub set of $v$, $\mathcal{C}$ updates the weight of the edge $(v, u)$ in $H$ to $w_H(v, u) = \min(w_G(v, u), \delta(v, u), \delta(u, v))$. (2) Whenever a vertex $u \in S^{(\tau-1)}(v) \setminus S^{(\tau)}(v)$ leaves the hub set of $v$, then if $(v, u) \in I$ or $u \in U$ and $v \in S^{(\tau)}(u)$, $\mathcal{C}$ updates the weight of the edge $(v, u)$ in $H$ to $w_H(v, u) = \min(w_G(v, u), \delta(u, v))$ (as $\delta(v, u) = \infty$ after the deletion), otherwise $\mathcal{C}$ removes $(v, u)$ from $H$. (3) Whenever a vertex $u \in S^{(\tau)}(v) \setminus S^{(\tau-1)}(v)$ enters the hub set of $v$, $\mathcal{C}$ adds the edge $(v, u)$ to $H$ (unless it exists already) and updates its weight to $w_H(v, u) = \min(w_G(v, u), \delta(v, u), \delta(u, v))$. Note that the number of these

[6] Note that $\tau$ is the number of updates processed only by $\mathcal{A}$ during the phase.

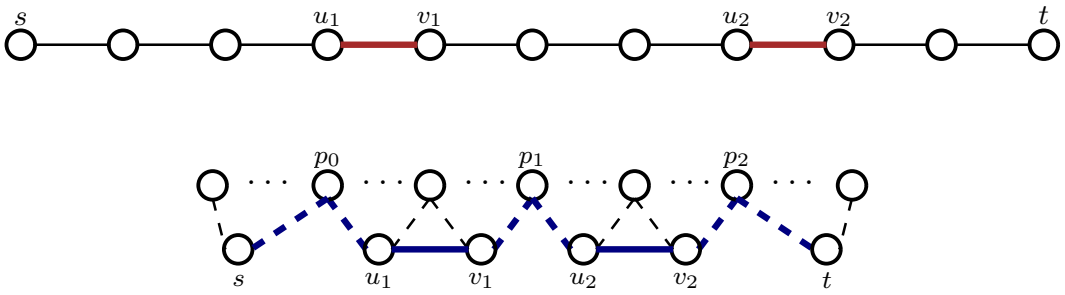



**Figure 1** Illustration of an $s$-$t$ shortest path. The brown thick edges have been inserted since the beginning of the phase. The corresponding subgraph of the auxiliary graph $H$ is also depicted (note that the vertices $s$ and $t$ are not necessarily part of $H$). The blue thick edges are the ones that participate in the correctness analysis of the query. Dashed edges depict edges inside the hub sets.

changes at time $\tau$ of $\mathcal{A}$ is equal to $\chi^{(\tau)}(v)$ for a vertex $v \in V$. Observe also that based on the two properties of $\mathcal{A}$, the number of vertices that participate in $H$ during a phase of length $\ell$ is at most $\ell \cdot (2 + 2\mu)$. Thus we can always find a "fresh" vertex (of degree 0) in $H$.

Finally, all the changes performed to $H$ are processed by the fully dynamic distance oracle $\mathcal{B}$ running on $H$, where edge weight changes are simulated by a deletion followed by a re-insertion.

Now a query for the approximate distance between any pair of vertices $s$ and $t$ is answered by returning:

$$\delta_{\mathcal{C}}(s,t) = \min\left(\min_{p \in S^{(\tau)}(s) \cap V(H), q \in S^{(\tau)}(t) \cap V(H)} \left(\delta(s,p) + \delta_{\mathcal{B}}(p,q) + \delta(t,q)\right), \delta_{\mathcal{A}}(s,t)\right).$$

Whenever $S^{(\tau)}(s) \cap V(H) = \emptyset$ or $S^{(\tau)}(t) \cap V(H) = \emptyset$, we let the inner term $\min(\cdot)$ to be $\infty$.

**Correctness.** To prove the correctness of this algorithm, we need to show that $\text{dist}_G(s,t) \leq \delta_{\mathcal{C}}(s,t) \leq \alpha\beta \cdot \text{dist}_G(s,t)$. The lower bound $\text{dist}_G(s,t) \leq \delta_{\mathcal{C}}(s,t)$ is immediate, since for each approximate distance returned by $\mathcal{C}$, the corresponding path uses edges from $G$ or distance estimates from the decremental approximate hub-labeling scheme which are never an underestimation of the real distance. To prove the upper bound, consider a shortest path $\pi$ from $s$ to $t$ in $G$, and let $G_{\mathcal{A}}$ be the graph maintained by $\mathcal{A}$ (i.e., the edge set of $G_{\mathcal{A}}$ is $E(G_{\mathcal{A}}) = F \setminus D$). If the path $\pi$ contains only edges from the set $F \setminus D$, then $\delta_{\mathcal{C}}(s,t) \leq \delta_{\mathcal{A}}(s,t) \leq \alpha \cdot \text{dist}_{G_{\mathcal{A}}}(s,t) = \alpha \cdot \text{dist}_G(s,t)$, and the claim follows. Otherwise, let $(u_1,v_1),\ldots,(u_j,v_j) \in I$ denote the edges of $\pi$ that have been inserted since the beginning of the current phase in order of appearance on $\pi$. Furthermore, let $p_0 \in S^{(\tau)}(s) \cap S^{(\tau)}(u_1)$ be the vertex that "certifies" $\delta_{\mathcal{A}}(s,u_1)$, that is, $\delta_{\mathcal{A}}(s,u_1) = \delta(s,p_0) + \delta(u_1,p_0)$. Similarly, let $p_j \in S^{(\tau)}(v_j) \cap S^{(\tau)}(t)$ be the vertex that "certifies" $\delta_{\mathcal{A}}(v_j,t)$, and for every $1 \leq i \leq j-1$, let $p_i \in S^{(\tau)}(v_i) \cap S^{(\tau)}(u_{i+1})$ be the vertex that "certifies" $\delta_{\mathcal{A}}(v_i,u_{i+1})$ (see Figure 1). These vertices must exist by the definition of an approximate hub-labeling scheme. Furthermore, by the construction of $H$, the edges $(u_1,p_0)$ and $(v_j,p_j)$ have been inserted to $H$, because $u_1 \in U$ and $p_0 \in S^{(\tau)}(u_1)$, and $v_j \in U$ and $p_j \in S^{(\tau)}(v_j)$ respectively. Hence, the vertices $p_0$ and $p_j$ belong to $V(H)$, and the sum $\delta(s,p_0) + \delta_{\mathcal{B}}(p_0,p_j) + \delta(t,p_j)$ participates in the inner term $\min(\cdot)$. Therefore to analyze the claimed upper-bound on the stretch, we proceed as follows:

$$
\begin{aligned}
\delta_{\mathcal{C}}(s,t) &\leq \delta(s,p_0) + \delta_{\mathcal{B}}(p_0,p_j) + \delta(t,p_j) \\
&\quad \text{(stretch guarantee of } \mathcal{B}\text{)} \\
&\leq \delta(s,p_0) + \beta \cdot \mathrm{dist}_H(p_0,p_j) + \delta(t,p_j) \\
&\quad \text{(triangle inequality)} \\
&\leq \delta(s,p_0) + \beta \cdot \mathrm{dist}_H(p_0,u_1) \\
&\quad + \sum_{1 \leq i \leq j-1} \beta \cdot (\mathrm{dist}_H(u_i,v_i) + \mathrm{dist}_H(v_i,p_i) + \mathrm{dist}_H(p_i,u_{i+1})) \\
&\quad + \beta \cdot (\mathrm{dist}_H(u_j,v_j) + \mathrm{dist}_H(v_j,p_j)) + \delta(t,p_j) \\
&\quad (\mathrm{dist}_H \leq w_H) \\
&\leq \delta(s,p_0) + \beta \cdot w_H(p_0,u_1) + \sum_{1 \leq i \leq j-1} \beta \cdot (w_H(u_i,v_i) + w_H(v_i,p_i) + w_H(p_i,u_{i+1})) \\
&\quad + \beta \cdot (w_H(u_j,v_j) + w_H(v_j,p_j)) + \delta(t,p_j)
\end{aligned}
$$

By the construction of $H$, the edges $(u_i, v_i)$ of $\pi$ and the corresponding edges $(p_{i-1}, u_i)$ and $(v_i, p_i)$ have been inserted to $H$[7], because $(u_i, v_i) \in I$, $u_i \in U$ and $p_{i-1} \in S^{(\tau)}(u_i)$, and $v_i \in U$ and $p_i \in S^{(\tau)}(v_i)$ respectively. Hence by the definition of $w_H(\cdot)$, we can replace $w_H(u_i, v_i)$ with $w_G(u_i, v_i)$, $w_H(p_{i-1}, u_i)$ with $\delta(u_i, p_{i-1})$ and $w_H(v_i, p_i)$ with $\delta(v_i, p_i)$. As a result, we have that (where $\alpha \geq 1$ and $\beta \geq 1$):

$$
\begin{aligned}
\delta_{\mathcal{C}}(s,t) &\leq \delta(s,p_0) + \beta \cdot \delta(u_1,p_0) + \sum_{1 \leq i \leq j-1} \beta \cdot (w_G(u_i,v_i) + \delta(v_i,p_i) + \delta(u_{i+1},p_i)) \\
&\quad + \beta \cdot (w_G(u_j,v_j) + \delta(v_j,p_j)) + \delta(t,p_j) \\
&\quad (\pi \text{ is a shortest path}) \\
&= \delta(s,p_0) + \beta \cdot \delta(u_1,p_0) + \sum_{1 \leq i \leq j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \delta(v_i,p_i) + \delta(u_{i+1},p_i)) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \delta(v_j,p_j)) + \delta(t,p_j) \\
&\leq \beta \cdot (\delta(s,p_0) + \delta(u_1,p_0)) + \sum_{1 \leq i \leq j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \delta(v_i,p_i) + \delta(u_{i+1},p_i)) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \delta(v_j,p_j) + \delta(t,p_j)) \\
&\quad \text{(definition of approximate hub-labeling scheme)} \\
&= \beta \cdot \delta_{\mathcal{A}}(s,u_1) + \sum_{1 \leq i \leq j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \delta_{\mathcal{A}}(v_i,u_{i+1})) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \delta_{\mathcal{A}}(v_j,t))
\end{aligned}
$$

From the stretch guarantee of $\mathcal{A}$, it holds that $\delta_{\mathcal{A}}(u,v) \leq \alpha \cdot d_{G_{\mathcal{A}}}(u,v)$ for any pair of vertices $u, v \in V$. For any two vertices $v_i, u_{i+1}$ from the previous sum, the subpath of $\pi$ from $v_i$ to $u_{i+1}$ uses edges only from the set $F \setminus D$, implying that $d_{G_{\mathcal{A}}}(v_i, u_{i+1}) = d_G(v_i, u_{i+1})$. The same argument holds for the pairs $s, u_1$ and $v_j, t$, thus it follows that:

$$
\begin{aligned}
\delta_{\mathcal{C}}(s,t) &\leq \alpha\beta \cdot \mathrm{dist}_G(s,u_1) + \sum_{1 \leq i \leq j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \alpha \cdot \mathrm{dist}_G(v_i,u_{i+1})) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \alpha \cdot \mathrm{dist}_G(v_j,t))
\end{aligned}
$$

[7] If $v_i = u_{i+1}$ then $p_i = v_i$, and so $w_H(v_i, p_i) = w_H(p_i, u_{i+1}) = 0$.

$$\leq \alpha\beta \cdot \text{dist}_G(s, u_1) + \sum_{1 \leq i \leq j-1} \alpha\beta \cdot (\text{dist}_G(u_i, v_i) + \text{dist}_G(v_i, u_{i+1}))$$
$$+ \alpha\beta \cdot (\text{dist}_G(u_j, v_j) + \text{dist}_G(v_j, t)) = \alpha\beta \cdot \text{dist}_G(s, t).$$

**Update and Query time.** To analyze the running times, consider a fixed phase of length $\ell$. During the first phase, the query time is $Q_{\mathcal{B}}(2\ell, \ell, W)$ and the amortized update is $t_{\mathcal{B}}(2\ell, \ell, W)$, as the initially empty graph $G$ can have at most $2\ell$ vertices and $\ell$ edges after $\ell$ updates. For the subsequent phases we proceed as follows. By the construction of $H$ and the two properties of $\mathcal{A}$, the graph $H$ has at most $\min(\ell(2+2\mu), n)$ vertices and $\ell(1+2\mu)$ edges during the phase, and the maximum edge weight in $H$ is $nW$ (the maximum distance in $G$).[8] Moreover by the first property we have that $|S^{(\tau)}(s) \cap V(H)| \leq \gamma$ and $|S^{(\tau)}(t) \cap V(H)| \leq \gamma$. Therefore the query time is equal to:

$$Q_{\mathcal{C}}(n, m, W) = Q_{\mathcal{A}}(n, m, W) + \gamma^2 \cdot Q_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW).$$

Let us now analyze the amortized update time. Since the total update time of $\mathcal{A}$ is $T_{\mathcal{A}}(n, m, W)$ and the amortized update time of $\mathcal{B}$ is $t_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW)$ during the phase, it remains to bound the total number of updates to $H$ per phase. Whenever an edge $e = (u, v)$ is inserted to $G$, we add to $H$ the two endpoints $u$ and $v$ together with their hub sets $S^{(\tau)}(u)$ and $S^{(\tau)}(v)$, and at most $1 + 2\gamma$ updates can occur to $H$. Until $(u, v)$ is deleted from $H$, every update to $H$ between $u$, $v$ and their hub sets modifies an entry of the distance estimate $\delta(u, \cdot)$ or $\delta(v, \cdot)$. By the definition of $\chi^{(\tau)}(\cdot)$, the number of entries of the distance estimates $\delta(u, \cdot)$ and $\delta(v, \cdot)$ that are modified at time $\tau$ of $\mathcal{A}$ is equal to $\chi^{(\tau)}(u) + \chi^{(\tau)}(v)$. Hence until $(u, v)$ is deleted from $H$, the total number of updates to $H$ between $u$, $v$ and their hub sets is equal to $2 \cdot (\sum_\tau \chi^{(\tau)}(u) + \sum_\tau \chi^{(\tau)}(v)) = 2 \cdot (X(u) + X(v))$,[9] which is at most $4\zeta$ based on the second property of Lemma 3. Moreover, when the edge $e$ is deleted from $G$, at most $1 + 2\gamma$ updates can occur to $H$. Therefore, the total number of updates to $H$ that correspond to an inserted edge in $G$, is at most $2 + 4\gamma + 4\zeta = 2 + 4\mu$ per phase. Since there can be at most $\ell$ inserted edges per phase, the total number of updates to $H$ during a phase is at most $\ell(2 + 4\mu)$. This implies that the total time for processing all updates during a phase is $T_{\mathcal{A}}(n, m, W) + t_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW) \cdot \ell(2+4\mu)$, which (when amortized over the $\ell$ updates of the previous phase) amounts to an amortized update time of:

$$T_{\mathcal{C}}(n, m, W) = \frac{T_{\mathcal{A}}(n, m, W)}{\ell} + t_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW) \cdot (2+4\mu). \quad \blacktriangleleft$$

## 3.2 Decremental approximate hub-labeling scheme

In this section, we argue that an existing decremental distance oracle from [35] also provides an approximate hub-labeling scheme whose properties make the reduction of Lemma 3 quite efficient. This decremental algorithm is based on the well-known static Thorup-Zwick (TZ) distance oracle [44].

**Thorup-Zwick distance oracle.** Given a graph $G = (V, E)$, the construction starts by defining a non-increasing sequence of sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$, where for each $1 \leq i < k$, the set $A_i$ is obtained by subsampling each element of $A_{i-1}$ independently with probability $n^{-1/k}$.

[8] We can assume that $\delta(\cdot, \cdot)$ is upper bounded by $nW$ whenever it has a finite value, since the maximum distance in $G$ is at most $nW$. Likewise, we can use the value $nW + 1$ instead of $\infty$.

[9] We multiply by 2 because edge weight changes are simulated by a deletion followed by a re-insertion.

For every vertex $v \in V$ and $1 \leq i < k$, let $\delta(v, A_i) = \min_{u \in A_i} \text{dist}_G(v, u)$ be the minimum distance from $v$ to a vertex in $A_i$. As $A_k = \emptyset$, we let $\delta(v, A_k) = \infty$. Moreover, let $p_i(v) \in A_i$ be a vertex in $A_i$ closest to $v$, that is, $\text{dist}_G(v, p_i(v)) = \delta(v, A_i)$. Then, the bunch $B(v) \subseteq V$ of each $v \in V$ is defined as:

$$B(v) = \bigcup_{i=0}^{k-1} B_i(v) \ , \ \text{ where } \ B_i(v) = \{u \in A_i \setminus A_{i+1} : \text{dist}_G(v, u) < \text{dist}_G(v, A_{i+1})\}.$$

The cluster of a vertex $u \in A_i \setminus A_{i+1}$ is defined as $C(u) = \{v \in V : \text{dist}_G(v, u) < \text{dist}_G(v, A_{i+1})\}$. Observe that $u \in B(v)$ if and only if $v \in C(u)$, for any $u, v \in V$.

As noted in [44], this construction is a hub-labeling scheme of stretch $2k-1$ (see Definition 2), where the hub set $S(v)$ of a vertex $v \in V$ is $S(v) = B(v) \cup (\bigcup_{i=0}^{k-1} \{p_i(v)\})$. In other words, bunches and pivots of all the $k$ levels form a hub set for $v$. For obtaining the distance estimates $\delta(v, \cdot)$ for all $v \in V$ as in Definition 2, we need the associated distances $\delta(v, u) = \text{dist}_G(v, u)$ for all $u \in S(v)$. It can be shown that with a simple modification of the stretch argument (e.g. see [29]), it is enough to only use the bunches as the hub sets, and explicit access to pivots is not necessary. Hence for simplifying the presentation in this section we assume that the hub sets are equivalent with the bunches. As shown in [44], the size of the bunch of any vertex is w.h.p. bounded by $\tilde{O}(n^{1/k})$. Recall that the maximum hub set size is one of the parameters governing the efficiency of our reduction.

In the next lemma we present the decremental algorithm of [35] which has good properties for the reduction of Lemma 3. For a more detailed explanation of the lemma see the full version [26].

▶ **Lemma 4** (Implicit in [35]). *Given a weighted undirected graph $G = (V, E)$ and $k > 1, 0 < \epsilon < 1$, there is a decremental hub-labeling scheme of stretch $(2k-1)(1+\epsilon)$ and w.h.p. the total update time is $\tilde{O}(mn^{1/k}) \cdot O(\log nW/\epsilon)^{2k+1}$. Moreover, w.h.p. we have the following two properties:*

1. *$\forall v \in V$ and $\forall \tau : |S^{(\tau)}(v)| \leq \tilde{O}(n^{1/k})$. In other words, the size of the bunch of any vertex is bounded by $\tilde{O}(n^{1/k})$ at any moment of the algorithm.*
2. *$\forall v \in V : X(v) \leq \tilde{O}(n^{1/k}) \cdot O(\log nW/\epsilon)^{2k+1}$. In other words, for every vertex $v \in V$ the total number of changes to $\delta(v, \cdot)$ is at most $\tilde{O}(n^{1/k}) \cdot O(\log nW/\epsilon)^{2k+1}$ over the course of the algorithm. Moreover the algorithm detects and reports these changes explicitly.*

## 3.3 Putting it together

In this section we explain how to obtain our final fully dynamic distance oracle by using the decremental algorithm of Section 3.2 in our reduction of Lemma 3.

▶ **Theorem 5.** *For any integer parameters $i \geq 0, k > 1$, there is a fully dynamic distance oracle $\mathcal{B}_i$ with stretch $(4k)^i$ and w.h.p. the amortized update time is $t_{\mathcal{B}_i}(n, m, W) = \tilde{O}(1)^{ki} \cdot m^{3/(3i+1)} \cdot n^{4i/k}$ and the query time $Q_{\mathcal{B}_i}(n, m, W) = \tilde{O}(1)^i \cdot n^{2i/k}$.*

**Proof.** The proof is by induction on the parameter $i$. For the base case $i = 0$, let $\mathcal{B}_0$ be the trivial fully dynamic distance oracle that achieves stretch 1, amortized update time $t_{\mathcal{B}_0}(n, m, W) = O(n^3)$, and query time $Q_{\mathcal{B}_0}(n, m, W) = O(1)$, by recomputing all-pairs shortest paths from scratch after each update (e.g., with the Floyd–Warshall algorithm).

For the induction step, let $\mathcal{A}$ denote the decremental approximate hub-labeling scheme from Lemma 4 with stretch $\alpha = 4k$ and w.h.p. total update time $T_{\mathcal{A}}(n, m, W) = \tilde{O}(1)^k \cdot mn^{1/k}$ and query time $Q_{\mathcal{A}}(n, m, W) = \tilde{O}(1) \cdot n^{1/k}$, where $\epsilon$ has been replaced with any value strictly smaller than $\frac{1}{2}$. By inductive hypothesis, we have that $\mathcal{B}_i$ (with $i \geq 0$) is a fully dynamic

distance oracle of stretch $\beta_i = (4k)^i$ with amortized update time $\tilde{O}(1)^{ki} \cdot m^{3/(3i+1)} \cdot n^{4i/k}$ and query time $\tilde{O}(1)^i \cdot n^{2i/k}$. Based on Lemma 4, w.h.p. the decremental approximate hub-labeling scheme $\mathcal{A}$ satisfies the properties of Lemma 3 with $\gamma = \tilde{O}(1) \cdot n^{1/k}$ and $\zeta = \tilde{O}(1)^k \cdot n^{1/k}$. By applying then Lemma 3 to $\mathcal{A}$ and $\mathcal{B}_i$ with $\ell = m^{(3i+1)/(3i+4)}$, the resulting fully dynamic distance oracle $\mathcal{B}_{i+1}$ has stretch $(4k)^{i+1}$, and amortized update time: $t_{\mathcal{B}_{i+1}}(n,m,W) = T_{\mathcal{A}}(n,m,W)/\ell + t_{\mathcal{B}_i}(n,\ell(1+2\mu),nW)\cdot(2+4\mu)$. The first term is equal to: $\tilde{O}(1)^k \cdot mn^{1/k}/\ell = \tilde{O}(1)^k \cdot m^{3/(3i+4)} \cdot n^{1/k} = \tilde{O}(1)^k \cdot m^{3/(3(i+1)+1)} \cdot n^{1/k}$, and the second term is equal to (where $\mu = \tilde{O}(1)^k \cdot n^{1/k}$):

$$
\begin{aligned}
&t_{\mathcal{B}_i}(n,\ell(1+2\mu),nW)\cdot(2+4\mu) = \tilde{O}(1)^{ki}\cdot(\ell\cdot\tilde{O}(1)^k\cdot n^{1/k})^{3/(3i+1)}\cdot n^{4i/k}\cdot\tilde{O}(1)^k\cdot n^{1/k}\\
&\text{(Replace } \ell \text{ with } m^{(3i+1)/(3i+4)})\\
&= \tilde{O}(1)^{ki}\cdot(m^{(3i+1)/(3i+4)}\cdot\tilde{O}(1)^k\cdot n^{1/k})^{3/(3i+1)}\cdot n^{4i/k}\cdot\tilde{O}(1)^k\cdot n^{1/k}\\
&\text{(Replace } n^{3/(3i+1)k} \text{ with } n^{3/k} \text{ and } \tilde{O}(1)^{3k/(3i+1)} \text{ with } \tilde{O}(1)^{3k})\\
&= \tilde{O}(1)^{ki}\cdot m^{3/(3i+4)}\cdot\tilde{O}(1)^{3k}\cdot n^{3/k}\cdot n^{4i/k}\cdot\tilde{O}(1)^k\cdot n^{1/k}\\
&= \tilde{O}(1)^{ki+k}\cdot m^{3/(3i+4)}\cdot n^{(4i+4)/k} = \tilde{O}(1)^{k(i+1)}\cdot m^{3/(3(i+1)+1)}\cdot n^{4(i+1)/k}.
\end{aligned}
$$

Therefore the amortized update time of $\mathcal{B}_{i+1}$ is:

$$t_{\mathcal{B}_{i+1}}(n,m,W) = \tilde{O}(1)^{k(i+1)}\cdot m^{3/(3(i+1)+1)}\cdot n^{4(i+1)/k}.$$

Finally the query time of $\mathcal{B}_{i+1}$ is (where $\gamma^2 = \tilde{O}(1)^2 \cdot n^{2/k}$):

$$
\begin{aligned}
Q_{\mathcal{B}_{i+1}}(n,m,W) &= Q_{\mathcal{A}}(n,m,W) + \gamma^2\cdot Q_{\mathcal{B}_i}(n,\ell(1+2\mu),nW)\\
&= \tilde{O}(1)\cdot n^{1/k} + \tilde{O}(1)^2\cdot n^{2/k}\cdot\tilde{O}(1)^i\cdot n^{2i/k} = \tilde{O}(1)^{i+1}\cdot n^{2(i+1)/k}.
\end{aligned}
$$

and so the distance oracle $\mathcal{B}_{i+1}$ has the desired guarantees. ◀

**Proof of Theorem 1.** By Theorem 5, for any $i \geq 1, k > 1$, there is a fully dynamic distance oracle $\mathcal{B}_i$ of stretch $(4k)^i$ that w.h.p. achieves $\tilde{O}(1)^{ki}\cdot m^{1/i}\cdot n^{4i/k}$ amortized update time and $\tilde{O}(1)^i \cdot n^{2i/k}$ query time. Since $m \leq n^2$, by setting $i = \frac{4}{\rho}$ and $k = \frac{64}{\rho^2}$ the claim follows. ◀

In Theorem 5, we can set $i$ to be a constant and set $k = O(\log\log n)^{1/i}$ to obtain another tradeoff, which is summarized in the following corollary.

▶ **Corollary 6.** *Given a weighted undirected graph $G = (V,E)$, there is a fully dynamic distance oracle with stretch $O(\log\log n)$ that w.h.p. achieves $n^{o(1)}$ query time and $\tilde{O}(n^\rho)$ amortized update time, for an arbitrarily small constant $\rho$.*

Finally note that we can also obtain similar tradeoffs as [25] where all three of stretch, amortized update time and query time are $n^{o(1)}$, by setting $k = O(\log\log n)^2$ and $i = O(\log\log n)$ in Theorem 5.

## References


1 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. *CoRR*, abs/2211.07058, 2022.

2 Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in P via short cycle removal: cycle detection, distance oracles, and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1487–1500, 2022. doi:10.1145/3519935.3520066.

3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2014)*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.

4 Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proc. of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 440–452, 2017. doi:10.1137/1.9781611974782.28.

5 Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the $o(n)$ barrier. In *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2014) and the 18th International Workshop on Randomization and Computation (APPROX/RANDOM 2014)*, pages 1–16, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.1.

6 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hierarchical hub labelings for shortest paths. In *Algorithms–ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings 20*, pages 24–35. Springer, 2012.

7 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.

8 Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *arXiv preprint arXiv:1607.05127*, 2016.

9 Aaron Bernstein. Fully dynamic $(2+\epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 693–702. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.16.

10 Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016. Announced at STOC 2013. doi:10.1137/130938670.

11 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 1000–1008, 2021. doi:10.1109/FOCS52979.2021.00100.

12 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proc. of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 365–376, 2019. doi:10.1145/3313276.3316335.

13 Jan Van Den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1011–1022, 2022. doi:10.1109/FOCS54457.2022.00099.

14 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *FOCS*, pages 436–455. IEEE Computer Society, 2019.

15 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *FOCS*, pages 456–480. IEEE Computer Society, 2019.

16 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In Mikkel Thorup, editor, *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 170–181, 2018. doi:10.1109/FOCS.2018.00025.

17 Shiri Chechik and Tianyi Zhang. Faster deterministic worst-case fully dynamic all-pairs shortest paths via decremental hop-restricted shortest paths. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 87–99. SIAM, 2023.

18 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1135–1146. IEEE, 2020. doi:10.1109/FOCS46700.2020.00109.

19 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In Samir Khuller and Virginia Vassilevska Williams, editors, *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 626–639. ACM, 2021. doi:10.1145/3406325.3451025.

20 Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In *Proceedings of the 55th ACM Annual Symposium on Theory of Computing (STOC 2023)*, 2023. arXiv:2304.09321.

21 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

22 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. Announced at STOC 2003. doi:10.1145/1039488.1039492.

23 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006. Announced at FOCS 2001. doi:10.1016/j.jcss.2005.05.005.

24 Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos. New tradeoffs for decremental approximate all-pairs shortest paths. *arXiv preprint arXiv:2211.01152*, 2022.

25 Sebastian Forster, Gramoz Goranci, and Monika Henzinger. Dynamic maintenance of low-stretch probabilistic tree embeddings with applications. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2021)*, pages 1226–1245, 2021. doi:10.1137/1.9781611976465.75.

26 Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping dynamic distance oracles, 2023. arXiv:2303.06102.

27 Sebastian Forster, Yasamin Nazari, and Maximilian Probst Gutenberg. Deterministic incremental APSP with polylogarithmic update time and stretch. *CoRR*, abs/2211.04217, 2022. doi:10.48550/arXiv.2211.04217.

28 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proc. of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2562–2574, 2020. doi:10.1137/1.9781611975994.156.

29 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1053–1072. SIAM, 2014.

30 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $o(mn)$ barrier and derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016. Announced at FOCS 2013. doi:10.1137/140957299.

31 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *Journal of the ACM*, 65(6):36:1–36:40, 2018. Announced at FOCS 2014. doi:10.1145/3218657.

32 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC 2015)*, pages 21–30, 2015. doi:10.1145/2746539.2746609.

33 Monika Rauch Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 664–672. IEEE, 1995.

**34** Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999. doi:10.1109/SFFCS.1999.814580.

**35** Jakub Łącki and Yasamin Nazari. Near-Optimal Decremental Hopsets with Applications. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

**36** Aleksander Mądry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 121–130, 2010. doi:10.1145/1806689.1806708.

**37** Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, (STOC 2010)*, pages 603–610, 2010. doi:10.1145/1806689.1806772.

**38** Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM Journal on Computing*, 37(5):1455–1471, 2008. Announced at FOCS 2002. doi:10.1137/060650271.

**39** Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. Announced at ESA 2004. doi:10.1007/s00453-010-9401-5.

**40** Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. Announced at FOCS 2004. doi:10.1137/090776573.

**41** Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *COCOON*, volume 3595 of *Lecture Notes in Computer Science*, pages 461–470. Springer, 2005.

**42** Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004)*, pages 384–396, 2004. doi:10.1007/978-3-540-27810-8_33.

**43** Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

**44** Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

# A Sweep-Plane Algorithm for Calculating the Isolation of Mountains

**Daniel Funke**
Karlsruhe Institute of Technology, Germany

**Nicolai Hüning**
Karlsruhe Institute of Technology, Germany

**Peter Sanders**
Karlsruhe Institute of Technology, Germany

## Abstract

One established metric to classify the significance of a mountain peak is its isolation. It specifies the distance between a peak and the closest point of higher elevation. Peaks with high isolation dominate their surroundings and provide a nice view from the top. With the availability of worldwide Digital Elevation Models (DEMs), the isolation of all mountain peaks can be computed automatically. Previous algorithms run in worst case time that is quadratic in the input size. We present a novel sweep-plane algorithm that runs in time $\mathcal{O}(n \log n + pT_{\text{NN}})$ where $n$ is the input size, $p$ the number of considered peaks and $T_{\text{NN}}$ the time for a 2D nearest-neighbor query in an appropriate geometric search tree. We refine this to a two-level approach that has high locality and good parallel scalability. Our implementation reduces the time for calculating the isolation of every peak on Earth from hours to minutes while improving precision.

**2012 ACM Subject Classification** Theory of computation → Theory and algorithms for application domains

**Keywords and phrases** computational geometry, Geo-information systems, sweepline algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.51

**Related Version** *Technical Report*: `https://arxiv.org/abs/2305.08470` [20]

**Supplementary Material** *Software*: `https://github.com/dfunke/mountains`
archived at `swh:1:dir:9081f9961a7408668c156e998a1facb21b93e7dc`

**Funding** This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

## 1 Introduction

High-resolution digital elevation models (DEMs) are an interesting example of large datasets with a big potential for applications but equally big challenges due to their enormous size. For example, WorldDEM provided by the TanDEM-X mission covers the entire globe with a resolution of 0.4 arcsecond and is currently the highest-resolution worldwide DEM available [39]. It consists of $6 \cdot 10^{12}$ individual sample points and amounts to approximately 25 TB of data. Modern LIDAR technology allows $< 1\,\text{m}^2$ samples, resulting in more than 300 TB of data for the land surface of the Earth. The algorithm engineering community has greatly contributed to unlocking the potential of DEMs by developing scalable algorithms for features such as contour lines, watersheds, and flooding risks [1, 11, 31]. This paper continues this line of research by studying the isolation of mountain peaks which is a highly nonlocal feature and requires us to deal with the Earth's complicated (not-quite spherical) shape.

A mountain's significance is typically characterized using three properties – elevation, isolation, and prominence [23]. Whereas elevation is a fundamental property, isolation and prominence are derived measures. Isolation – also referred to as dominance – measures the



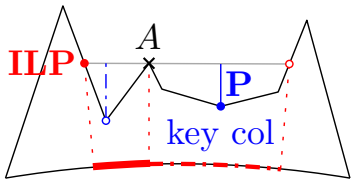




**Figure 1** Illustration of a mountain $A$'s isolation (red) and prominence (blue). The isolation is measured along the sea-level surface of the Earth.

distance along the sea-level surface of the Earth between the peak and the closest point of higher elevation, known as the *isolation limit point* (ILP). Prominence measures the minimum difference in elevation from a peak and the lowest point on a path to reach higher ground – called the *key col*. Refer to Figure 1 for an illustration.

Whereas previously both measures had to be determined manually by laboriously studying topographic maps, they can now be computed algorithmically using DEMs. Kirmse and de Ferranti [28] present the current state-of-the-art to compute both measures. They use an algorithm to determine a peak's isolation by searching in concentric rectangles of increasing size around the peak for higher ground. To determine the isolation of every peak on Earth, their algorithm has a worst case running time of $\mathcal{O}(n^2)$, with $n$ being the number of sample points in the DEM. For high-resolution DEMs of the Earth and other celestial bodies, e. g. the Moon [6] or Mars [24], algorithms with better scalability are needed. Also, Continuously Updated Digital Elevation Models (CUDEMs), such as NOAA's DEM of North America's coastal regions [2], require efficient algorithms for frequent reprocessing. In particular, parallel and external algorithms are required.

**Contribution and Outline.** After presenting basic concepts in Section 2 and related work in Section 3 we describe the main algorithmic result in Section 4. We begin with a sequential algorithm that sweeps the surface of the Earth top-down with a sweep-plane storing the points having surfaces at that elevation. Processing a peak then amounts to a nearest neighbor query in the sweep-plane data structure. This results in an algorithm with running time $\mathcal{O}(n \log n + pT_{\text{NN}})$ where $n$ is the input size, $p$ the number of considered peaks and $T_{\text{NN}}$ the time for a 2D nearest-neighbor query in an appropriate geometric search tree. To make this more scalable, we then develop an algorithm working on the natural hierarchy of the data which is specified in *tiles*. This algorithm performs most of its work in two scans of the data which can work independently and in parallel tile-by-tile. Only the highest points in each tile need to be processed in an intermediate global phase. Each of these three phases has a structure similar to the simple sequential algorithm.

In Section 5 we then explain how 2D-geometric search trees can be adapted to work on the surface of the Earth by deriving the required geometric predicates. After outlining implementation details in Section 6, in Section 7 we evaluate our approach using the largest publicly available DEM data.

## 2 Preliminaries

**Spherical Geometry.** Planets can generally be approximated by spheres. In the geographic coordinate system, a point $p = (\phi, \lambda)$ on the surface of a sphere is identified by its latitude $\phi$ and longitude $\lambda$. Latitude describes $p$'s north-south location, measured from the equator, $\phi \in [-90°, 90°]$ with negative values south of the equator. Longitude describes $p$'s east-west location, measured from the prime meridian through Greenwich, $\lambda \in (-180°, 180°]$ with negative values west of the prime meridian.

For close distances, Earth's surface is sufficiently flat to use planar Euclidean distances between points. For longer distances, spherical distance calculations are necessary, which are computationally more expensive. On the surface of the sphere, two points are always connected by at least two great circle segments. The shortest such segment, the *geodesic*, can be computed for points $p_1 = (\lambda_1, \phi_1)$ and $p_2 = (\lambda_2, \phi_2)$ with sphere radius $R$ according to

$$\begin{aligned} \alpha &= \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \sin^2\left(\frac{\lambda_1 - \lambda_2}{2}\right)\cos(\lambda_1)\cos(\lambda_2) \\ d(p_1, p_2) &= R\tan^{-1}\left(\frac{\sqrt{\alpha}}{\sqrt{(1-\alpha)}}\right) \end{aligned} \tag{1}$$

As the Earth and most other planets are not perfect spheres, ellipsoids are a more accurate approximation of their shape. The World Geodetic System (WGS) [29] defines such an ellipsoidal approximation of the Earth, requiring the following – even more computationally expensive – distance formula:

$$\begin{aligned} \Delta\lambda &= (\lambda_2 - \lambda_1)/2 \qquad \Delta\phi = (\phi_2 - \phi_1)/2 \qquad \Lambda = (\lambda_2 + \lambda_1)/2 \\ s &= \sin^2\Delta\phi \cdot \cos^2\Delta\lambda + \cos^2\Lambda \cdot \sin^2\Delta\lambda \\ c &= \cos^2\Delta\phi \cdot \cos^2\Delta\lambda + \sin^2\Lambda \cdot \sin^2\Delta\lambda \\ w &= \tan^{-1}(\sqrt{s}/\sqrt{c}) \qquad r = \frac{\sqrt{sc}}{w} \\ d(p_1, p_2) &= 2aw\left(1 + f\frac{3r-1}{2c}\sin^2\Lambda \cdot \cos^2\Delta\phi - f\frac{3r+1}{2s} \cdot \sin^2\Delta\phi \cdot \cos^2\Lambda\right) \end{aligned} \tag{2}$$

with $a$ and $f$ denoting the equatorial radius and flattening of the WGS84 ellipsoid [29].

**Digital Elevation Models.** Digital Elevation Models (DEMs) have become one of the most important tools to analyze the Earth's surface in geographic information systems. They represent the surface of the Earth by providing elevation measurements on a grid of sample points. The data is mostly stored in files of 1 square degree of coverage each, called tiles. A tile is addressed by its smallest latitude and longitude. In order to enable seamless processing of several tiles, each tile stores one sample row/column overlap with its neighbors. The resolution of DEMs is given by the length of one sample at the equator in arcseconds ($''$).

State-of-the-art worldwide DEMs, such as WorldDEM, provide a resolution of $0.4''$ spacing between sample points – about 12 m at the equator [39]. Local DEMs, such as the swissALTI$^{3D}$, even have a resolution of only 0.5 m sample point spacing [21]. Freely available DEMs, such as The Shuttle Radar Topography Mission (SRTM) [25], provide a resolution of $3''$ – about 90 m at the equator – and an absolute vertical height error of no more than 16 m for 90 % of the data [32]. Unfortunately, it does not provide global coverage[1] and contains large void areas, especially in mountainous regions, which are of particular interest for us. In a laborious process, de Ferranti [12] fused raw SRTM data with other publicly available datasets [14, 36] and digitized topographic maps to create a worldwide, void-free DEM available at `www.viewfinderpanoramas.org`.

[1] Only areas between 60° North and 56° South are covered.

## 3 Related work

Graff et al. [22] are the first to use DEM data to classify terrain into mounts, plains, basins or flats. As there is no definitive definition of these terrain features, several methods for terrain classification are studied in the literature [38], including fuzzy logic [16, 17] or, more recently, deep learning [37].

Prominence has received most of the attention when it comes to algorithmic computation of mountain metrics [26, 28]. Kirmse and de Ferranti [28] present the current state-of-the-art regarding isolation and prominence computation. As their main focus lies on the prominence calculation, they present a rather simple $\mathcal{O}(n^2)$ time isolation calculation algorithm.

In their algorithm they first calculate potential peaks which are samples that are at least as high as their eight neighboring samples. Afterwards, a search for the closest higher ground (ILP) is conducted, where, centered on each peak, concentric rectangles of increasing size are checked to find a sample with higher elevation. Since the closest higher ground of a peak could also be on a neighboring tile, these might be checked as well. If a higher ground was found, the distance to this sample is used to constrain the search in neighboring tiles. If not, tiles in increasing rectangles around the peak-tile are checked until an ILP is found or the complete world has been checked. Before searching a neighboring tile, the maximum elevation of the tile is checked. When it is smaller than the peak elevation, the tile can be ignored. Tiles and their maximum elevation are cached, because they need to be loaded rather frequently. Inside the tile that contains the peak, planar Euclidean distance approximations are used to find an ILP, for neighboring tiles, distances are computed according to the spherical distance function. Because a majority of peaks have a small isolation, the ILP is often within the same tile as the peak, so mostly planar Euclidean distance approximations are used to determine a peak's ILP. This reduces computational costs but also the accuracy for peaks with small isolation. Only in a final step before output is the distance between a peak and its determined ILP calculated using the precise, but expensive, ellipsoid distance function. Kirmse and de Ferranti's algorithm is unnamed, we refer to it as ConcIso for brevity.

Sweepline algorithms are introduced by Bentley and Ottmann [7] to compute line segment intersections. The technique is generalized by Anagnostou et al. to three-dimensional space [3]. Sweepline algorithms have been applied to various geometric problems in two- and three-dimensional space, such as computing Voronoi diagrams [18] or route mining [40]. R-Trees are often used in spatial databases and have been adapted to geodetic distance computations by Schubert et al. [33].

## 4 Algorithm

In this section we present our novel sweep-plane algorithm to calculate the isolation of mountain peaks. The algorithm takes as input the search area – a quadrilateral $A$ defined by its north-west and south-east corners – as well as the DEM data. We will first present a single-sweep algorithm that processes the entire search area in one sweep-plane pass. Subsequently, we present a scalable three-pass algorithm that reads the DEM-tiles of $A$ twice and can process tiles in parallel.

### 4.1 Single-Sweep Algorithm

To determine the isolation of a peak, its closest point with higher elevation – the Isolation Limit Point (ILP) – needs to be found. Given the search area $A$, each sample point $p$ of the DEM within $A$ has two associated events: an *insert* event at $p$'s elevation and a *remove* event at the elevation of its lowest immediate (NESW) neighbor. Additionally, if a point is

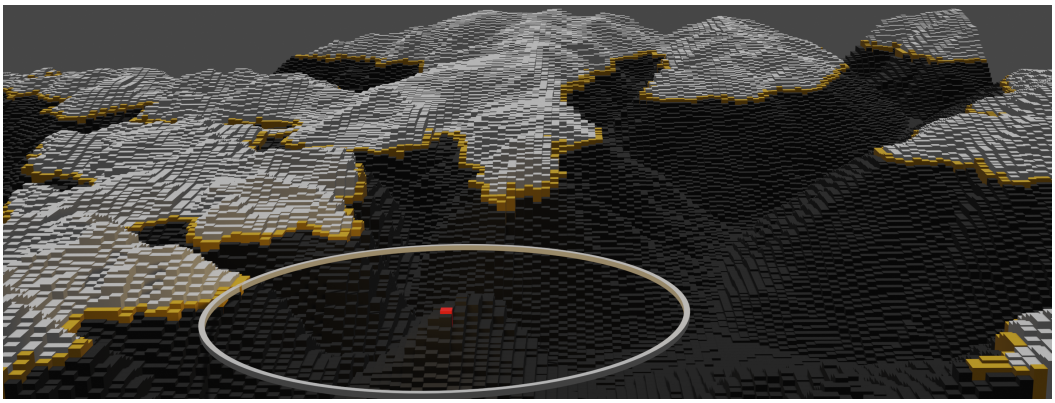


**Figure 2** Illustration of DEM grid and sweep-plane algorithm by using a voxel representation. Red represents the current peak. Orange DEM points are active and contained in the sweep-plane data structure. White points are inactive and already removed from the data structure. Black points have not yet been processed.

the high-point of its 3x3 neighborhood it is associated with a *peak* event at its elevation. We describe the peak detection routine in more detail in Section 6. All events are created at the beginning of the algorithm and can therefore be added to a static sequence sorted by descending elevation. Peak events are processed before other events at the same elevation.

The sweep-plane then moves downward from the highest sample point to the lowest and traces the contour lines of the terrain, refer to Figure 2. The sweep-plane data structure $SL$ is a two-dimensional geometric search tree that maintains a set of currently *active* points. A sample point $p$ becomes active at its insert event, when it is swept by the sweep-plane and inserted into $SL$. Point $p$ becomes *inactive* and is removed from $SL$ at its remove event, at which point its lowest neighboring point is activated and thus all of $p$'s neighboring points are either active or have already been deactivated. When a peak event for sample point $p$ is processed, a nearest neighbor query for the closest active sample point to $p$ in $SL$ is performed. The returned point $w$ is the ILP for the peak: Since $w$ is active and peaks are processed before other events, $w$ must be higher than $p$. There cannot be any closer ILP as points activated later are not higher than $p$ and since higher points $v$ that are already deactivated are surrounded by points that are all higher than $p$. At least one of them must be closer to $p$ than $v$.

**Analysis.** Given a DEM with $n$ points, we can detect its $p$ peaks in time $\mathcal{O}(n)$. The resulting $2n + p$ events can be sorted in $\mathcal{O}(n \log n)$ time by elevation. Insertion and removal operations take $\mathcal{O}(\log n)$ worst case time in several geometric search trees [8, 15, 35]. Nearest neighbor search complexity is more complicated and still an open problem in many respects. Therefore, we describe it abstractly as $T_{\text{NN}}$. Simple trees used in practice such as $k$-D-Trees [8] or Quadtrees [15] achieve logarithmic time "on average". Cover trees [9] achieve logarithmic time when an *expansion parameter* of the input set is bounded by a constant. Approximate queries within a factor of $1 + \epsilon$ are possible in time $\mathcal{O}\big(\log(n)/\epsilon^2\big)$ [4]. Overall, we get the claimed time $\mathcal{O}(n \log n + pT_{\text{NN}})$.

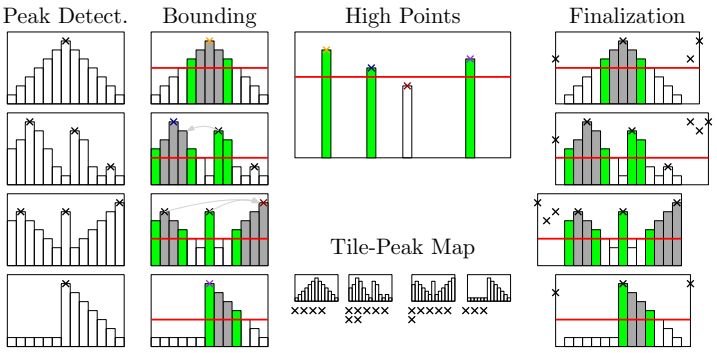



**Figure 3** Overview of our scalable multi-pass algorithm for four tiles in two dimensions. Peaks are detected for each tile with the peak finding algorithm of ConcIso. The bounding pass determines an upper bound on the isolation for each peak and assigns it to tiles that could contain closer ILPs in the Tile-Peak map. The high-points of each tile without a local upper bound are processed by a separate pass considering only the high-points. In the finalization pass, all peaks assigned to a tile are checked for an ILP within the tile. The final determination of the closest ILP out of the found candidates for each peak is not depicted. For the sweep-plane passes, green points are active in the sweep-plane data structure, gray points are inactive and hollow points are not yet processed. Peaks are marked by crosses. The high-point of each tile is marked by a different color cross for each tile.

## 4.2 Scalable Multi-Pass Algorithm

High-resolution DEMs are massive data sets of currently up to 25 TB that require scalable algorithms. The algorithm described in the previous section can be parallelized to some extent but its sweeping character limits parallelism. Moreover, a geometric search tree covering the entire Earth could get quite large. We therefore develop a two-level algorithm that allows for more coarse-grained parallelism and better locality. We adopt the natural hierarchy of the input data using tiles of a fixed area but note that reformatting into smaller or finer tiles would be possible in principle. Furthermore, a more general multi-level algorithm could be developed using a similar approach.

Multi-level algorithms start at the finest level to extract information for global[2] processing at coarser level. The global results are then passed down to compute the final solution. In our two-level algorithm this implies that we have two passes reading the DEM tiles from external memory while a single global pass works with simple per-tile information. The first (*bounding*) and last (*finalization*) pass can work in parallel on each tile. The global (*high-point*) pass works in internal memory and is also parallelized. The first two passes establish a global Tile-Peak map that stores for each tile which peaks *can* have an ILP in it. The third pass processes these assigned peaks for a tile and determines their ILPs. All passes follow a very similar structure to our single-sweep algorithm from Section 4.1. They are described in detail in the following. Additionally, Figure 3 provides an overview of our algorithm.

**Bounding Pass.** The purpose of the first pass is to establish an upper bound on the isolation of a peak and therefore limit the the number of tiles that need to be searched for its ILP in the finalization pass. To establish this upper bound, we find a tile-local ILP for each peak using our single-sweep algorithm. Only the highest point in each tile will not have a tile-local ILP and is treated separately in the high-point pass. Given the upper bound on the isolation

[2] Pun intended.

of a peak $p$, we can assign $p$ to all tiles within radius of the upper bound that could contain a closer ILP for $p$. In the third pass, these neighboring tiles then need to process $p$ in order to find ILP candidates within them. To link peaks to tiles for processing in the third pass, we build a Tile-Peak map that stores for each tile a list of peaks that could have an ILP within it.

**High-Point Pass.** As there is no local ILP for the high-point $h$ of each tile, we address high-points separately in the second pass. This pass uses only one type of event – a combination of insert and peak event from the other passes – and processes only one point per tile, namely their high-points. For each high-point $h$, we perform a nearest neighbor query in the sweep-plane data structure for $h$'s closest higher point $p$. The distance between $h$ and $p$ is an upper bound on $h$'s isolation. The sweep-plane data structure is then traversed again to find all tiles whose closest point to $h$ is at least as close to $h$ as $p$ and $h$ is linked to these tiles in the Tile-Peak map. Finally, $h$ is inserted into the sweep-plane data structure and the next lower high-point is processed.

**Finalization Pass.** In the third pass, each tile is processed again by a sweep-plane algorithm similar to the single-sweep variant. The peak events of this pass are all peaks that have been assigned to the tile in the Tile-Peak map. For each assigned peak, the ILP candidate within the processed tile is determined. After all tiles have been processed by the third pass, each peak has as many ILP candidates as tiles it has been linked to. Thus, in a final step, for each peak the closest found ILP candidate is set as the true ILP of the peak.

**Algorithmic Details and Outline of Analysis.** Let us first look at the total **work** performed to process $A$ tiles containing $n$ sample points overall. The bounding pass performs a similar amount of work as the global algorithm except that it defers nonlocal nearest neighbor searches to the subsequent passes. More precisely, the high-points of each tile are deferred to the high-point pass while other peaks are deferred to neighboring tiles. On average, there is a constant number of such neighbor tiles.[3]

The high-point pass potentially defers nearest-neighbor-search work for the highest point of each tile to a potentially large number of tiles. However, this is not so different from what a search-tree based nearest neighbor search in a global tree data structure would do. Our two-level algorithm can be viewed as a vertically split (quad-)tree algorithm where updates and nearest neighbor searches are reordered to improve locality. The overall amount of work done is quite similar though.

The finalization pass can be viewed as completing the deferred nearest neighbor searches. Thus, the main overhead of the two-pass algorithm compared to the global algorithm is that the data is swept through search trees twice. We mitigate this effect by building only a coarse search tree in the bounding pass. This has no negative effect on precision as the bounding pass is only needed to identify the tiles where an ILP can be. Only the finalization pass computes the actual ILPs.

Let us now look at **I/O-costs**. Assume one tile and the high-points fit in internal memory. This is similar to a "semi-external"-assumption used in previous algorithm engineering papers on DEM processing, e. g. [1]. The I/O volume of our two pass algorithm is dominated by

[3] Near the poles, there are many candidate tiles that may be closer than a local ILP. However, averaged over all tiles, this effect is not large. It is interesting to note though that this is an artifact of a pseudo-high longitudinal resolution that is not reflected in the actual precision of the sensors but stems from artificially mapping the data using a Mercator projection. Meshes with more uniform cells are possible, for example the icosahedral grid used in some modern climate/weather models [27].

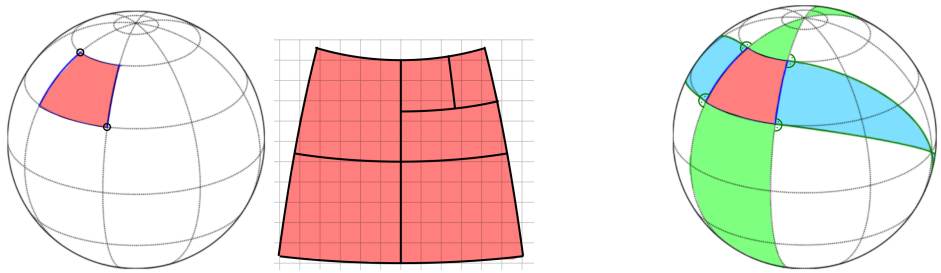



**Figure 4** Representation of a latitude-longitude aligned quadrilateral and areas for the different min-distance cases between a quadrilateral and a point.

reading all tiles twice.[4] This is actually better than an external-memory implementation of the global algorithm as that algorithm would have to sort the data by altitude before being able to scan the input in the right order for the sweep. Even using pipelining [13], sorting would require at least two reading and one writing pass over the DEM data.

Finally let us consider **parallelization**. Bounding and finalization passes can work on $A$ tiles in parallel. In order to also parallelize the high-point pass, we implement a variant that uses a static search tree with subtrees augmented by the maximum elevation occuring in a subtree. Then the $A$ nearest-dominating-point searches can be done in parallel. See Section 6 for details.

## 5 Predicates for Search Trees on Spherical Surfaces

The sweep-plane data structure needs to be a dynamic data structures that supports efficient nearest neighbor queries. Space-partitioning trees such as $k$-D trees or Quadtrees are well-suited data structures for this application [8, 15]. These trees recursively divide the input space into smaller and smaller blocks. For a spherical surface, the space is divided into quadrilaterals which are aligned with latitude and longitude, refer to Figure 4a. A quadrilateral is defined by its north-west and south-east corners. Each quadrilateral can then be further subdivided into smaller quadrilaterals. The root of a space-partitioning tree covers the entire input area.

Our sweep-plane algorithm requires two geometric primitives for these trees: a) whether a given point $p$ lies inside a quadrilateral $Q$ and b) the shortest distance between $p$ and $Q$. The former can be easily answered by comparing $p$'s latitude and longitude with $Q$'s north-east and south-west corner. For the latter, there are four configurations of $p$ and $Q$ that need to be considered, refer to Figure 4b:

1. $p \in Q$ (red area),
2. $p$ is between the longitude lines of $Q$ (green area),
3. $p$ is between the four great circles through the corners of $Q$, which are perpendicular to the longitude edges of $Q$ (blue area),
4. all other positions (white area).

[4] In comparison, the Tile-Peak map has negiligible data volume and can be handled in an I/O-efficient way by observing that the first two passes only insert to it and the third pass only reads it tile by tile. Thus, we can first buffer its data in an external log which is sorted by tile before the finalization pass.

For case 1 the distance is zero. In case 2, the closest point $s \in Q$ to $p$ is on the intersection between the longitude line of $p$ and one of the latitude-edges of the quadrilateral, as the shortest distance between two latitudes is along the longitude lines. The shortest distance of $p$ to $Q$ is thus the shorter distance between $p$ and the north and south latitude of $Q$.

In the remaining cases, $s$ must lie on one of the longitude lines of the quadrilateral. To determine the longitude edge closest to $p$, we calculate the center longitude of $Q$ and rotate it to align with the meridian. We rotate $p$ by the same amount. If the longitude of $p$ is now positive, the west longitude-edge of $Q$ is closer to $p$, otherwise the east one.[5] Having determined the closest longitude edge, we can now calculate point $s$ using linear algebra in Euclidean space as described in the accompanying technical report [20, Appendix A.2]. If the latitude of $s$ is between the top and bottom latitude of $Q$, $s$ is the point with the shortest distance to $p$ in $Q$ (case 3). Otherwise one of the corners is the point with the shortest distance to $p$ (case 4).

Given these primitives, insert and query operations on space-partitioning trees for spherical surfaces are identical to the ones for Euclidean space.

## 6 Implementation

We implement our new sweep-plane algorithm in the MOUNTAINS C++ framework by Kirmse and de Ferranti [28].[6] The framework provides essential functionalities for the work with tiled DEM data, such as data loading and conversion, peak discovery and distance computations. Our code is available on Github.[7] In the following we provide details our implementation.

**Data Structure.** We implement a fully dynamic $k$-D-Tree, that supports insert and remove operations in $\mathcal{O}(\log n)$ worst case time and nearest neighbor searches in $\mathcal{O}(\log n)$ expected time [19] using the predicates described in the previous section.[8] Points are stored in the leaves of the tree, which have a fixed capacity of $C$ points. On exceeding capacity $C$, a leaf is split according to a center-split policy along the longer side of the quadrilateral. The first points inserted into the tree often belong to the highest peak in a tile and are thus close to each other. In order to prevent the tree from degenerating, we pre-build the first $k$ levels in a Quadtree-like manner.[9] To improve cache-efficiency, tree nodes are allocated in blocks and are re-used after deletion.

Another data structure used is the Tile-Peak map, which we implement as an internal memory hash map with the latitude and longitude of a tile as key and a list of peaks as value.

**Algorithm.** Given the search area $A$, all contained DEM tiles can be processed in parallel. We use a work queue and thread pool to distribute the computations among the available processing elements. The Tile-Peak map is initialized in advance with all tiles. To sort the events of a pass, we use the efficient sorting algorithm ips4o of Axtmann et al. [5].

[5] This is possible because we split the Earth at the antimeridian. Therefore, the western longitude of $Q$ is always smaller than the eastern one.
[6] `https://github.com/akirmse/mountains`
[7] `https://github.com/dfunke/mountains`
[8] We also adapted a Quadtree using our predicates, which was however outperformed by the $k$-D-Tree in our experiments.
[9] Our experiments show $k = 4$ to be a good choice.

**Bounding Pass.** We use the peak detection algorithm of Kirmse and de Ferranti [28, Sec. 2.2] to find all peaks within a tile. It considers all points to be peaks that have eight neighboring points of lower or equal elevation. If a peak consists of several equally high sample points, only a single one is added to the set of peaks.[10] Since only an upper bound is calculated during the bounding pass, we down-sample the resolution of the DEM after peak detection. Since all peaks are within the tile which is processed, distances are rather small and can be approximated using planar Euclidean geometry during the nearest-neighbor search. The upper bound is computed using the spherical distance function according to Equation (1). If the determined upper bound on the isolation of a peak is below a threshold $I_{\min}$ we discard the peak as insignificant. Peaks with an upper bound above $I_{\min}$ are added to the Tile-Peak map for processing in the finalization pass as described in Section 4.2.

**High-Point Pass.** While processing the tiles in the bounding pass, we build a geometric search tree $T$ that partitions the entire search area down to the tile level. Internal nodes of $T$ save the highest elevation in its sub-tree. After all tiles have been processed, we can use this information to efficiently determine upper bounds on the isolation of the high-points of each tile. For each high-point $h$, we find the closest tile containing a point of higher elevation than $h$ in search tree $T$. The maximum distance between $h$ and any point within the found tile serves as an upper bound on $h$'s isolation. Given this upper bound we can add $h$ to all tiles containing potential ILPs in the Tile-Peak map. This approach is trivially parallelizable over the number of high-points in the search area.

**Finalization Pass.** In this pass we use the full resolution of the input DEM. For each tile, the peaks processed in this pass are the ones that are assigned to it in the Tile-Peak map. We use ellipsoid distance computations according to Equation (2).

## 7 Evaluation

In this section we evaluate our novel sweep-plane algorithm to calculate the isolation of mountain peaks, which we named SweepIso. We evaluate it with regard to runtime behavior and solution quality and compare it against ConcIso from Kirmse and de Ferranti [28].

[10] The algorithm by Kirmse and de Ferranti always chooses the north-west corner of a peak.

**Table 1** DEM models of the Earth, Mars and the Moon used in our experiments. The reported runtime is the single-threaded runtime for the entire data set.

| Name | Resolution | Pixels | Size | Coverage | Runtime |
|---|---|---|---|---|---|
| Earth SRTM | $3''$ (90 m) | $20 \times 10^9$ | 71 GB | Global | 2.07 h |
| Earth SRTM NA-EU | $1''$ (30 m) | $49 \times 10^9$ | 105 GB | Cont. US+CAN+EU | 3.8 h |
| Moon SLDEM2015 | $7''$ (59 m) | $11 \times 10^9$ | 22 GB | $[60°N, 60°S]$ | 2.5 h |
| Mars MGS MOLA - MEX HRSC Blended | $12''$ (197.6 m) | $5.7 \times 10^9$ | 11 GB | Global | 1.3 h |

**Experimental Setup.** All benchmarks are conducted on a machine with an AMD EPYC Rome 7702P with 64 cores and 1024 GB of main memory. The DEM data is stored on a Intel P4510 2 TB NVMe SSD. We use the 3″ data set from viewfinderpanoramas [12] with worldwide coverage. To build smaller test instances from the worldwide data set, we choose a random starting tile and add neighboring tiles in a spiraling manner around it until the desired number of tiles is reached. For each instance size, we generate several instances to cover a wide range of terrain. Additionally, we use the 3″ and 1″ DEM for the USA, Canada and most of Europe from viewfinderpanoramas [12] to study the scaling behavior for higher-resolution DEMs. This data set corresponds to 4294 tiles or roughly 16 % of the total test data set. We will call this data set NA-EU. We also use data sets from other celestial bodies: Mars [24] and the Moon [6]. Table 1 lists some properties of the studied data sets along with the runtime of our algorithm to determine the isolation of every peak contained in them.

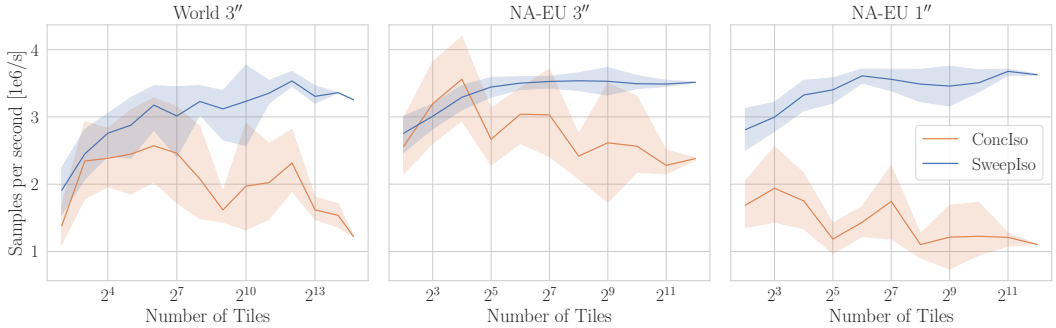
For all benchmarks we use an isolation threshold $I_{\min}$ of 1 km and report the mean runtime of 5 runs. I/O costs are not part of the reported figures as we use the MOUNTAINS framework [28] for them without an attempt at optimization. They are about the same time as computation for 3″ DEMs and about 10 % of computation time for 1″ ones and thus could be overlapped with the computation in an optimized framework.

## 7.1 Runtime and Scaling Behavior

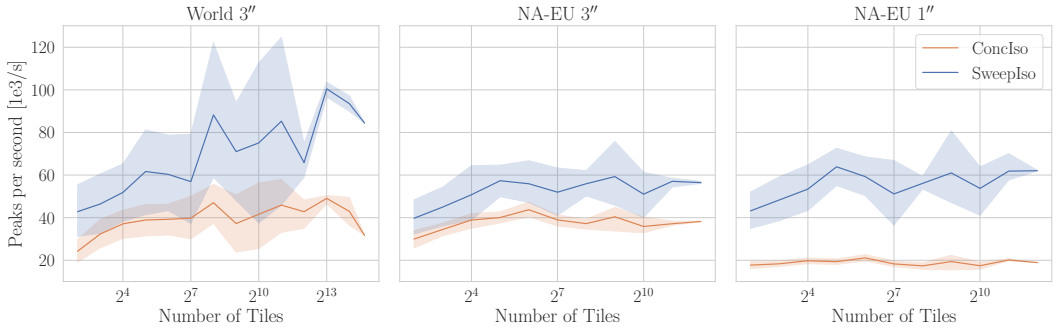
The runtime of the algorithms depends on the number of sample points in the DEM. These can either increase due to a larger search area or a higher-resolution DEM. Another factor is the number of processed peaks, since every peak starts a local search in ConcIso and a nearest neighbor query in SweepIso. A larger search area increases the number of tiles and the number of peaks, whereas higher-resolution data mostly increases the number of points per tile. High-resolution DEMs can contain more peaks than lower-resolution ones due to the more truthful representation of the terrain, however these are predominantly low-isolation peaks and are filtered out in the first pass. We study both effects in our experiments by using different resolution DEMs as well as increasing search areas.

SweepIso exhibits a nearly constant throughput of sample points per second with increasing search area, while ConcIso's throughput degrades – refer to Figure 5a. Figure 8a shows that SweepIso outperforms ConcIso by a factor of 2 to 3 in terms of sample point throughput. For instance, we reduce the time required to compute the isolation of every peak on Earth from 9 h down to 3.5 h. However, ConcIso's runtime scales significantly better than its $\mathcal{O}(n^2)$ worst case runtime bound would suggest. This is because most peaks have a relatively low isolation. In fact 99.996 % of discovered peaks on the world data set have an isolation below 50 km and more than 99 % below 10 km. Since one tile covers on average an area of 70 km × 111 km, the nearest higher point is most of the time within the same tile as the peak, where fast approximations for the distance calculation are used. Nevertheless, for high-resolution DEMs the computation cost per peak is significantly higher for ConcIso than for SweepIso, refer to Figure 5b. A more in-depth analysis of the runtime behavior of both algorithms can be found in the accompanying technical report [20, Appendix B].

Figure 6 shows the runtime composition of the bounding and the finalization pass. As the DEM resolution is reduced in the bounding pass, the finalization pass requires the majority of the runtime, especially for higher-resolution inputs. Geometric computations only make a small fraction of the runtime. The high-point pass requires just 0.001 % of the total execution time and is therefore omitted in the figure.




**(a)** Throughput in sample points per second over number of tiles.




**(b)** Throughput in processed peaks per second over number of tiles.

**Figure 5** Single-threaded runtime comparison of SweepIso and ConcIso.

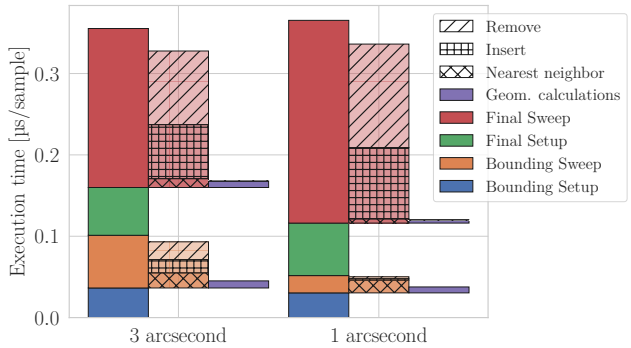



**Figure 6** Runtime composition of the bounding and finalization pass for 3″ and 1″ data. The sweep-plane operations of these passes are further subdivided into remove, insert, nearest neighbor search operations as well as the time for distance calculations.

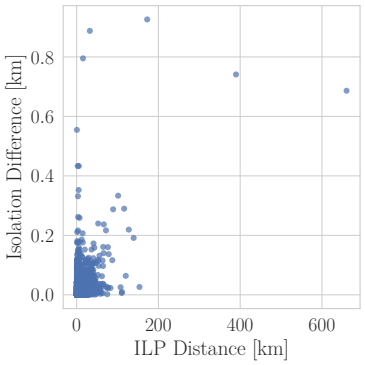
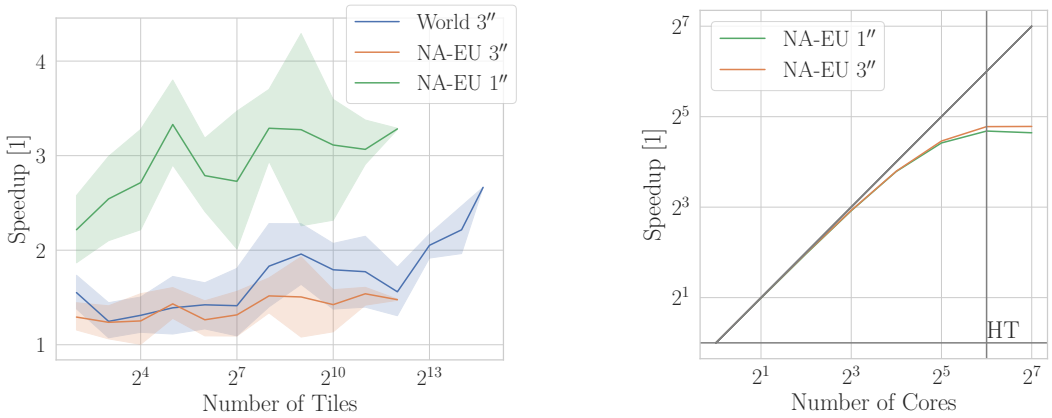



**Figure 7** Comparison of difference in isolation between SweepIso and ConcIso and distance between found ILPs.




**(a)** Single-threaded speedup.

**(b)** Multi-threaded relative speedup.

**Figure 8** Single-threaded speedup of SweepIso over ConcIso and relative speedup of SweepIso for multi-threaded execution.

Figure 8b shows the results of our multi-threaded runtime experiments with the NA-EU data set. SweepIso scales well with physical cores, but does not benefit from hyper-threading (HT). For 64 cores it reaches a speedup of 25. We verified the scaling behavior of the algorithm for the entire Earth and were able to confirm a speedup of 25 on 64 cores. This corresponds to a runtime of 8 min to calculate the isolation of every peak on Earth. We were not able to execute ConcIso with multiple threads due to issues in the implementation.

## 7.2 Solution Quality

In principle, at any particular time, the isolation and witnessing ILP of a peak are a well defined. However, the actually computed values depend on imprecisions in both the data and the used algorithms.

Due to the design of SweepIso, more expensive distance approximations can be used to find the ILP than in ConcIso. This results in closer ILPs being found, as shown in Figure 7, which displays the distribution of the difference in isolation and the distance between the found ILPs between SweepIso and ConcIso, using the same data. Even if the isolation values between the two algorithms do not vary greatly, the distances between the ILPs do. For example for the Cerro Gordo summit in Mexico both algorithms find ILPs that are more than 600 km apart while SweepIso's ILP is merely 0.8 km closer to the peak.

To further evaluate the results, we used the collection of peaks with more than 300 km isolation from the website `peakbagger.com` [34]. The comparison showed that for about 75 % of peaks in this list the isolation deviation is below 2 km. In a DEM, a sample point corresponds to the average elevation of the area it represents. This often leads to an underestimation of a peak's elevation, sometimes significantly [28]. For instance, according to our DEM data, Galdhøpiggen (the highest point of Scandinavia) is 11 m lower than Glittertind. This causes a significant change in isolation of both mountains. Table 2 lists the five summits with the biggest differences in isolation between peakbagger and our calculation. The five most isolated peaks on Mars and the Moon as determined by our algorithm are presented in Table 3. To the best of our knowledge, this is the first such list.

**Table 2** Biggest isolation differences between discovered and peakbagger (PB) data.

| Mountain | PB Rank | PB Isolation | SweepIso Isolation | Notes |
|---|---|---|---|---|
| Galdhøpiggen (Norway) | 47 | 1568.3 | 12.9 | *Galdhøpiggen*<br>DEM elev.: 2455 m – real elev.: 2469 m [30]<br>*Glittertind*<br>DEM elev.: 2466 m – real elev.: 2457 m [30]<br>Isolation 1563.5 km |
| Mt. Kirkpatrick (Antarctica) | 45 | 1585.2 | 53.5 | found ILP at -83.8967;168.3733<br>DEM elev.: 4416 m – real elev.: 4528 m [30]<br>ILP elev.: 4416 m |
| Mt. Hope (Antarctica) | 75 | 1113.5 | 203.4 | No data found in other sources |
| Dome Charlie (Antarctica) | 92 | 971.8 | 248.6 | found ILP at -76.3783;116.3708<br>DEM elev.: 3265 m – real elev.: 3233 m [10]<br>ILP elev. 3266 m |
| Mauga Silisili (Samoa) | 23 | 2245.1 | 2502.8 | DEM elev.: 1854 m – real elev.:1863 m [30]<br>PB ILP elev.: 1879 m<br>found ILP elev.: 1836 m |
| Cocos Islands High Point | 94 | 961.4 | 947.4 | Found ILP at Enggano Island |

**Table 3** Most isolated peaks on Mars and the Moon according to the DEM data listed in Table 1.

| Name | Coordinates | | Elevation | ILP | | Isolation |
|---|---|---|---|---|---|---|
| *Mars* | | | | | | |
| Olympus Mons | 17.33°N | 133.42°W | 21 226 m | | - | $\infty$ |
| Cruls crater wall | 42.27°S | 163.78°E | 4331 m | 46.79°S | 147.63°W | 2037.43 km |
| Near Cyclopia | 6.34°S | 129.19°E | 3806 m | 18.59°N | 149.99°E | 1913.59 km |
| Huygens crater wall | 10.06°S | 54.62°E | 4604 m | 10.50°S | 85.09°E | 1776.76 km |
| Ascraeus Mons | 11.76°N | 104.53°W | 18 321 m | 17.89°N | 131.64°W | 1593.92 km |
| *The Moon* | | | | | | |
| Engel'gardt crater wall | 5.41°N | 158.63°W | 10 783.3 m | | - | $\infty$ |
| Between Tacitus and Fermat craters | 18.93°S | 19.17°E | 4832.33 m | 60.0°S | 72.65°W | 2261.5 km |
| Near Lewis crater | 21.04°S | 112.33°W | 9459.83 m | 5.26°N | 157.98°W | 1574.56 km |
| Near Calippus crater | 39.09°N | 9.46°E | 3625.68 m | 9.63°S | 2.06°E | 1492.13 km |
| Dellinger crater wall | 7.25°S | 142.0°E | 7561.73 m | 12.91°S | 170.22°W | 1434.95 km |

## 8 Conclusion and Future Work

We have presented SweepIso, a scalable and efficient algorithm to compute the isolation of peaks. SweepIso considerably outperforms the previous, more brute-force, state-of-the-art approach. The performance gains also enable more accurate distance calculations at decisive places resulting in higher accuracy. SweepIso is able to process the entire Earth for currently publicly available data within minutes. This is relevant as it indicates that SweepIso can also handle higher-resolution data that is available commercially or that will be available in the future. SweepIso's two-level semi-external sweeping architecture may also be an interesting design pattern for other computations on massive DEM data. Furthermore, SweepIso could serve as a benchmark for dynamic nearest neighbor search data structures.

**Future Work.** From an application perspective, it would be interesting to compute not only isolations for a given, necessarily imprecise data set but to compute confidence bounds that take into account error margins in the input data. This would be possible at a moderate increase in cost. Peaks could be replaced by enclosing boxes/circles while vertical errors could be handled by having "may-be-there" and "must-be-there" insertion events and sweep-plane data structures. Geographically most interesting would be those isolations that change a lot depending on how high exactly particular pairs of peaks are. Those pairs could then be valuable targets for additional data cleaning or new measurements.

For algorithm engineering, it would be interesting to close the gap between theory and practice with respect of nearest-neighbor data structures. We have reasonable empirical performance of simple data structures like $k$-D trees but no well-fitting performance guarantees applicable to SweepIso. For example, one could look for a more general characterization of inputs where cover trees [9] work provably well.

## References


1 Pankaj K. Agarwal, Lars Arge, Thomas Mølhave, and Bardia Sadri. I/O-efficient efficient algorithms for computing contours on a terrain. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*. ACM, June 2008. doi:10.1145/1377676.1377698.

2 Christopher J. Amante, Matthew Love, Kelly Carignan, Michael G. Sutherland, Michael MacFerrin, and Elliot Lim. Continuously Updated Digital Elevation Models (CUDEMs) to Support Coastal Inundation Modeling. *Remote Sensing*, 15(6), 2023. doi:10.3390/rs15061702.

3 Efthymios G. Anagnostou, Leonidas J. Guibas, and Vassilios G. Polimenis. Topological sweeping in three dimensions. In Tetsuo Asano, Toshihide Ibaraki, Hiroshi Imai, and Takao Nishizeki, editors, *Algorithms*, pages 310–317, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

4 Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

5 M. Axtmann, S. Witt, D. Ferizovic, and P. Sanders. In-Place Parallel Super Scalar Samplesort (IPSSSSo). In *25th Annual European Symposium on Algorithms*, volume 87, pages 9:1–9:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2017.9.

6 M.K. Barker, E. Mazarico, G.A. Neumann, M.T. Zuber, J. Haruyama, and D.E. Smith. A new lunar digital elevation model from the Lunar Orbiter Laser Altimeter and SELENE Terrain Camera. *Icarus*, 273:346–355, 2016. doi:10.1016/j.icarus.2015.07.039.

7 J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, pages 643–647, 1979.

8 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975. doi:10.1145/361002.361007.

9 Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1143844.1143857.

10 COMNAP: Council of Managers of National Antarctic Programs. Antarctic Station Catalogue, 2017. Online; accessed 22-Apr-2023. URL: https://www.comnap.aq/s/COMNAP_Antarctic_Station_Catalogue.pdf.

11 Mark de Berg and Constantinos Tsirogiannis. Exact and approximate computations of watersheds on triangulated terrains. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, November 2011. doi:10.1145/2093973.2093985.

12 Jonathan de Ferranti. Digital elevation data, 2011. Online; accessed 05-Juli-2022. URL: http://viewfinderpanoramas.org/dem3.html.

13 R. Dementiev, L. Kettner, and P. Sanders. STXXL: Standard Template Library for XXL data sets. *Software Practice & Experience*, 38(6):589–637, 2008.

14 Alaska Satellite Facility. Radarset antarctic mapping, 2008. URL: https://asf.alaska.edu/data-sets/derived-data-sets/radarsat-antarctic-mapping-project-ramp/.

15 R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974. doi:10.1007/bf00288933.

16 Peter Fisher and Jo Wood. What is a mountain? or the englishman who went up a boolean geographical concept but realised it was fuzzy. *Geography*, pages 247–256, 1998.

17 Peter Fisher, Jo Wood, and Tao Cheng. Where is helvellyn? fuzziness of multi-scale landscape morphometry. *Transactions of the Institute of British Geographers*, 29(1):106–128, 2004.

18 S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987.

19 Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

20 Daniel Funke, Nicolai Hüning, and Peter Sanders. A Sweep-plane Algorithm for Calculating the Isolation of Mountains. Technical report, Karlsruhe Intitute of Technology, May 2023. doi:10.48550/arXiv.2305.08470.

21 Bundesamt für Landestopografie swisstopo. *swissALTI 3D Das hoch aufgelöste Terrainmodell der Schweiz*, March 2022.

22 L. H. Graff and E. L. Usery. Automated classification of generic terrain features in digital elevation models. *Photogrammetric Engineering and Remote Sensing*, 59(9):1409–1417, 1993.

23 Peter Grimm. *Die Gebirgsgruppen der Alpen: Ansichten, Systematiken und Methoden zur Einteilung der Alpen*. Deutscher Alpenverein, 2004.

24 K. Gwinner, F. Scholten, F. Preusker, S. Elgner, T. Roatsch, M. Spiegel, R. Schmidt, J. Oberst, R. Jaumann, and C. Heipke. Topography of Mars from global mapping by HRSC high-resolution digital terrain models and orthoimages: Characteristics and performance. *Earth and Planetary Science Letters*, 294(3-4):506–519, 2010.

25 Heather Hanson. Shuttle radar topography mission (srtm). Technical report, NASA, 2019. Online; access 21-september-2022. URL: https://eospso.gsfc.nasa.gov/missions/shuttle-radar-topography-mission.

26 Adam Helman. *The Finest Peaks-Prominence and Other Mountain Measures*. Trafford Publishing, 2005.

27 Johann H Jungclaus, Stephan J Lorenz, Hauke Schmidt, Victor Brovkin, Nils Brüggemann, Fatemeh Chegini, Traute Crüger, Philipp De-Vrese, Veronika Gayler, Marco A Giorgetta, et al. The ICON earth system model version 1.0. *Journal of Advances in Modeling Earth Systems*, 14(4):e2021MS002813, 2022.

28 Andrew Kirmse and Jonathan de Ferranti. Calculating the prominence and isolation of every mountain in the world. *Progress in Physical Geography*, 41(6):788–802, 2017.

**29** National Center for Geospatial Intelligence Standards. World geodetic system 1984: Its definition and relationships with local geodetic systems. Technical Report NGA.STND.0036_1.0.0_WGS84, Department of Defense, 2014.

**30** PeakVisor. Peakvisor. Online; accessed 22-Apr-2023. URL: `https://peakvisor.com/panorama.html`.

**31** Mathias Rav, Aaron Lowe, and Pankaj K. Agarwal. Flood risk analysis on terrains. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, November 2017. `doi:10.1145/3139958.3139985`.

**32** E Rodriguez, CS Morris, JE Belz, EC Chapin, JM Martin, W Daffer, and S Hensley. An assessment of the SRTM topographic products. Technical Report JPL D-31639, Jet Propulsion Laboratory, 2005.

**33** Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. Geodetic distance queries on r-trees for indexing geographic data. In *Advances in Spatial and Temporal Databases*, pages 146–164. Springer, Berlin Heidelberg, 2013. `doi:10.1007/978-3-642-40235-7_9`.

**34** Greg Slayden. Peakbagger. Online; accessed 22-Apr-2023. URL: `https://peakbagger.com`.

**35** Michiel Smid. Closest-point problems in computational geometry. In *Handbook of computational geometry*, pages 877–935. Elsevier, 2000.

**36** Tetsushi Tachikawa, Masami Hato, Manabu Kaku, and Akira Iwasaki. Characteristics of ASTER GDEM version 2. In *2011 IEEE International Geoscience and Remote Sensing Symposium*, pages 3657–3660, 2011. `doi:10.1109/IGARSS.2011.6050017`.

**37** Rocio Nahime Torres, Piero Fraternali, Federico Milani, and Darian Frajberg. A deep learning model for identifying mountain summits in digital elevation model data. In *IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 212–217. IEEE, 2018.

**38** Rocio Nahime Torres, Federico Milani, and Piero Fraternali. Algorithms for mountain peaks discovery: a comparison. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 667–674, 2019.

**39** Dimitra I. Vassilaki and Athanassios A. Stamos. TanDEM-X DEM: Comparative performance review employing LIDAR data and DSMs. *ISPRS Journal of Photogrammetry and Remote Sensing*, 160:33–50, 2020. `doi:10.1016/j.isprsjprs.2019.11.015`.

**40** Wang Yitao, Yang Lei, and Song Xin. Route mining from satellite-AIS data using density-based clustering algorithm. *Journal of Physics: Conference Series*, 1616(1):012017, August 2020. `doi:10.1088/1742-6596/1616/1/012017`.

# A Tight Competitive Ratio for Online Submodular Welfare Maximization

**Amit Ganz**
The Henry and Marilyn Taub Faculty of Computer Science, Technion, Haifa, Israel

**Pranav Nuti**
Department of Computer Science, Stanford University, CA, USA

**Roy Schwartz**
The Henry and Marilyn Taub Faculty of Computer Science, Technion, Haifa, Israel

## Abstract

In this paper we consider the online Submodular Welfare (SW) problem. In this problem we are given $n$ bidders each equipped with a general non-negative (not necessarily monotone) submodular utility and $m$ items that arrive online. The goal is to assign each item, once it arrives, to a bidder or discard it, while maximizing the sum of utilities. When an adversary determines the items' arrival order we present a simple randomized algorithm that achieves a tight competitive ratio of $1/4$. The algorithm is a specialization of an algorithm due to [Harshaw-Kazemi-Feldman-Karbasi MOR'22], who presented the previously best known competitive ratio of $3 - 2\sqrt{2} \approx 0.171573$ to the problem. When the items' arrival order is uniformly random, we present a competitive ratio of $\approx 0.27493$, improving the previously known $1/4$ guarantee. Our approach for the latter result is based on a better analysis of the (offline) Residual Random Greedy (RRG) algorithm of [Buchbinder-Feldman-Naor-Schwartz SODA'14], which we believe might be of independent interest.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online Algorithms, Submodular Maximization, Welfare Maximization, Approximation Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.52

**Related Version** *Full Version*: https://arxiv.org/abs/2308.07746

**Funding** Amit Ganz and Roy Schwartz have received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 852870-ERC-SUBMODULAR.

**Acknowledgements** The authors would like to thank the anonymous referees for helpful remarks.

## 1 Introduction

Submodularity is a mathematical notion that captures the concept of diminishing returns. Formally, a set function $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ over a ground set $\mathcal{N}$ is submodular, if for all $A, B \subseteq \mathcal{N}$: $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$. An equivalent definition, which is called the diminishing returns property, is the following: $f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B)$, for every $A \subseteq B \subseteq \mathcal{N}$ and every $u \in \mathcal{N} \setminus B$. Submodular functions naturally arise in many different settings, e.g., combinatorics, graph theory, information theory and economics.

We consider the Submodular Welfare (SW) problem. In this problem we are given a set $\mathcal{N} = \{1, \ldots, m\}$ of $m$ unsplittable items and a set $B = \{1, \ldots, n\}$ of $n$ bidders. Each bidder $j$ has a non-negative (and not necessarily monotone) submodular utility function $f_j$ and the goal is to assign items to the bidders while maximizing the sum of the utilities: $\sum_{j=1}^{n} f_j(S_j)$. Here $S_j$ is the set of items allocated to bidder $j$, and the requirement is that $S_j \cap S_{j'} = \emptyset$ for every $j \neq j'$ (since the items are unsplittable) and $\cup_{j=1}^{n} S_j \subseteq \mathcal{N}$ (note that not all items must be assigned). SW with monotone utilities has been extensively studied for more than two decades, e.g., [13, 15, 19, 25, 43, 52, 62, 68, 69]. Submodular maximization

with general (not necessarily monotone) objectives is also the focus of extensive theoretical research, e.g., [24, 30, 32, 34, 36, 50, 51, 63, 68, 70]. Moreover, maximization of non-monotone submodular objectives has found numerous practical applications, e.g., network inference [39], mobile crowdsensing [55], summarization of documents and video [53, 54, 59], marketing in social networks [64], and even gang violence reduction [65], to name a few. In particular, non-monotone utilities in the context of SW can model soft budget constraints, where each bidder $j$ needs to pay a price $p_{i,j}$ when item $i$ is allocated to it. Additional related problems were also studied, including: SW with demand queries [26], and other utilities such as XOS and subadditive [18, 23].

When considering the offline version, SW is typically viewed as a special case of maximizing a submodular objective subject to a partition matroid independence constraint: (1) the ground set is $\mathcal{N} \times B$; (2) the partition matroid is defined over $\mathcal{N} \times B$ where each part in the partition corresponds to an item, *i.e.*, the part that corresponds to item $i \in \mathcal{N}$ is $\{(i,j)\}_{j=1}^{n}$; and (3) the objective $f : 2^{\mathcal{N}\times B} \to \mathbb{R}_{\geq 0}$ is defined as: $f(S) \triangleq \sum_{j=1}^{n} f_j(\{i : (i,j) \in S\})$. In case the utilities are monotone a tight (asymptotic) approximation of $(1 - 1/e)$ was given by [14] by introducing the celebrated continuous greedy algorithm, a tight approximation of $(1 - (1 - 1/n)^n)$ for any number $n$ of bidders was given by [31], and the matching hardness result was given by [60]. For a general and not necessarily monotone objective the current best known offline approximation is also based on the continuous approach and achieves an approximation of $(1/e) + 0.0171$ [8], which improved upon the previous works of [21, 31].

In the online version of SW items arrive one by one. Whenever an item arrives, one has to decide immediately and irrevocably whether to assign it to one of the bidders or not assign it at all (the latter decision is relevant only when the utilities are not necessarily monotone). There are two natural settings that differ in the order in which items arrive. First, in the online adversarial setting an adversary can choose the order in which items arrive (the adversary knows the algorithm and how it operates, however if the algorithm is randomized it does not know the outcome of its random choices). Second, in the online random order setting items arrive one by one in a uniform random order.

When considering the online version, as opposed to the offline version, worse results are known. For monotone utilities, in the adversarial setting, a $(1/2)$-competitive greedy algorithm is known and additionally it is the best possible algorithm for this setting [41]. However, if one assumes the online random order setting, it is known that one can achieve a competitive ratio of $(1/2) + 0.0096$ [9], which improved the result of [46] who were the first to break the $1/2$ barrier obtaining a competitive ratio of $(1/2) + 0.005$.

Unfortunately, when considering general submodular (and not necessarily monotone) utilities, worse results are known. In the adversarial setting, the special case of a single bidder is of particular interest since it is equivalent to online Unconstrained Submodular Maximization (USM): given a general submodular objective $f$ items arrive one by one in an online manner and once an item arrives the algorithm needs to decide whether to choose or discard it where the goal is to maximize $f(S)$ ($S$ denotes the chosen elements). An (implicit) competitive ratio of $1/4$ was given by [24] for online USM (the algorithm simply chooses every element independently with a probability of half). This result was proved to be tight [11]. For the general case of multiple bidders an algorithm achieving a competitive ratio of $3 - 2\sqrt{2} \approx 0.171573$ was given by [38] (this algorithm handles a general matroid independence constraint and assumes elements of the ground set arrive in an online manner).

In the random order setting, an (implicit) result follows from analyzing the offline Residual Random Greedy (RRG) algorithm of [10] for maximizing a non-monotone submodular objective subject to a matroid independence constraint. When considering SW, the RRG algorithm

operates as follows: it chooses a uniform random order over the items and goes over the items in this order, and assigns each item to the bidder with the highest marginal value (if this marginal value is negative then the item is discarded). Thus, if the RRG algorithm achieves an approximation of $\alpha$ in the offline setting, then the deterministic greedy achives a competitive ratio of $\alpha$ in the online random order setting. [10] prove that the RRG algorithm achieves an approximation of $1/4$, therefore implying a competitive ratio of $1/4$ for SW in the random order setting. It is worth noting that better algorithms than RRG are known in the offline setting, e.g., [8, 21, 31], however, they do not apply to the online random order setting.

In this work, we assume the standard *value oracle* model: each submodular function $f$ is not given explicitly but rather the algorithm can query for every subset $S$ the value of $f(S)$. The running time of the algorithm is measured not only by the number of arithmetic operations, but also by the number of value queries. In the online version, for both adversarial and random order settings, the algorithm can query subsets $S$ only of items that already arrived. Thus, intuitively, the algorithm has no information regarding future items.

## 1.1 Our Results

Focusing first on the adversarial setting, we present the following positive result.

▶ **Theorem 1.** *There exists a randomized polynomial time algorithm achieving a competitive ratio of $1/4$ for online SW in the adversarial setting with general (not necessarily monotone) utilities.*

There are three things to note regarding Theorem 1. First, the competitive ratio of $1/4$ is tight as even for the special case of a single bidder (which is equivalent to online USM) [11] provide a matching hardness of $1/4$. Second, Theorem 1, to the best of our knowledge, improves the previous best known competitive ratio of $3 - 2\sqrt{2} \approx 0.171573$ [38]. Third, for the special case of a single bidder the competitive ratio of Theorem 1 matches the $1/4$ guarantee of the simple algorithm that just chooses independently for every item to include it in the solution with a probability of half [24]. However, we note that the algorithm we present in order to prove Theorem 1, even in the special case of a single bidder, differs from the algorithm that just chooses a uniform random subset.

We complement the above result by showing that the randomness of the online algorithm in Theorem 1 is needed. This is summarized in the following theorem that gives a hardness result that tends to zero.

▶ **Theorem 2.** *For every $M > 0$, no deterministic algorithm can achieve a competitive ratio better than $1/M$ for online SW in the adversarial setting with general (not necessarily monotone) utilities.*

Focusing on the random order setting, the following theorem proves that one can achieve an improved competitive ratio over the previously (implicit) known $1/4$ [10]. We note that the following theorem separates the random order and adversarial settings, since one should recall there is a hardness of $1/4$ in the adversarial setting even when only a single bidder is present.

▶ **Theorem 3.** *The deterministic greedy algorithm achieves a competitive ratio of $\approx 0.27493$ for online SW in the random order setting with general (not necessarily monotone) utilities.*

The above theorem is achieved by a better analysis of the offline randomized RRG algorithm of [10]. This improved analysis can be easily extended to a general matroid independence constraint, as the following theorem states (its proof is deferred to a full version of the paper).

▶ **Theorem 4.** *The RRG algorithm of [10] achieves an approximation guarantee of* $\approx 0.27493$ *for maximizing a general (not necessarily monotone) submodular function given a matroid independence constraint.*

## 1.2 Our Approach

The online algorithm for the adversarial setting adopts a simple randomized approach: once item $i$ arrives it defines a distribution over the bidders and assigns the item to a random bidder sampled from this distribution. Surprisingly, we prove that a remarkably simple distribution suffices to obtain a tight competitive ratio of $1/4$: item $i$ is assigned to the bidder with the highest marginal value with a probability of $1/2$, to the bidder with the second highest marginal with a probability of $1/4$, and so forth as long as the marginal value is non-negative. With the remainder probability item $i$ is discarded. It is important to note that the *ordering* of the bidders according to their marginal values (as long as the marginal values are non-negative) dictates the distribution. However, as long as the ordering remains the same, the distribution is independent of the actual marginal values themselves. We prove that the above simple approach suffices to obtain a tight competitive ratio of $1/4$ for the adversarial setting.

It should be noted that the above randomized approach is based on the streaming algorithm of [38], which obtains a competitive ratio of $3 - 2\sqrt{2} \approx 0.171573$. Intuitively, we specialize the algorithm of [38] to online SW in the adversarial setting. The reason is that once item $i$ arrives one can perform the following process in order to obtain the distribution over bidders that was defined above: sort the bidders in a non-increasing order of marginal values and assign the item to the first bidder with a probability of half, if the item was not assigned to this bidder then with a probability of half assign the item to the next bidder, and so forth as long as the marginal value is non-negative. This process describes how the algorithm of [38] operates in the special case the online order of elements of the ground set $\mathcal{N} \times B$ satisfies: (1) all elements $\{(i,j)\}_{j=1}^{n}$ are consecutive in the online order; and (2) elements $\{(i,j)\}_{j=1}^{n}$ are sorted in a non-increasing order of marginal values of the bidders.

Focusing on the random order setting, we present an improved analysis of the RRG algorithm for a general matroid independence constraint. As a preliminary step, which is not required but is mathematically convenient, we present a "smooth" version of the RRG algorithm. The difference between the original and smooth versions is that in the smooth version the distribution of the steps the algorithm can perform does not change as the algorithm progresses. However, in the original version this distribution evolves as the algorithm progresses. We note that the analysis of both versions is very similar (assuming the smooth version performs enough steps). Nonetheless, we believe it is easier to analyze the smooth version.[1]

A key insight in analyzing the RRG algorithm (as well as other closely related algorithms, e.g., the Random Greedy algorithm of [10]), is lower bounding the expected value of a fixed optimal solution $OPT$ when a random subset of elements $S$ is added to it. More precisely, if each element $u$ satisfies $\Pr[u \in S] \leq p$ then standard known arguments, e.g., Lemma 2.2 [10] which is based on Lemma 2.2 [24], imply that $\mathbb{E}[f(OPT \cup S)] \geq (1-p)f(OPT)$. It is important to note that this general insight works for *any* distribution of $S$ that satisfies $\Pr[u \in S] \leq p$ for every $u$. This general insight by itself enabled [10] to prove that the RRG algorithm achieves an approximation of $1/4$. We are able to improve upon the above

[1] For a detailed comparison of the two versions we defer to a full version of the paper.

by exploiting the specific probabilistic behavior of the smooth version of the algorithm, as opposed to the general insight which works for any distribution of $S$. This results in an improved analysis of the smooth version of the RRG algorithm via two jointly related recursive relations that together bound the performance of the algorithm.

## 1.3 Additional Related Work

The literature regarding submodular maximization, *i.e.*, problems of the form $\max\{f(S) : S \in \mathcal{F}\}$ ($\mathcal{F}$ is the collection of feasible solutions), is rich, e.g., [4, 7, 8, 10, 14, 21, 22, 24, 44, 47, 48, 60, 61, 66], and dates back to the late 70's. Moreover, the online SW problem naturally generalizes many online problems such as online matching [1, 29, 40, 42, 57], online weighted matching [1, 28], budgeted allocation [12, 16, 35, 58], and more general classes of online allocation problems [2, 17, 27, 67]. Other related problems include the extensively studied class of secretary problems, e.g., [3, 5, 6, 20, 31, 33, 37, 45, 49, 56], where the goal is to solve an optimization problem assuming the arrival order of the input is uniform and random.

## 1.4 Preliminaries

In our analysis we require the following known lemma.

▶ **Lemma 5** (Lemma 2.2 [10] which is based on Lemma 2.2 [24]). *Let $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ be submodular. Denote by $A(p)$ a random subset of $A$ where each element appears with probability at most $p$ (not necessarily independently). Then $\mathbb{E}[f(A(p))] \geq (1-p)f(\emptyset)$.*

## 1.5 Paper Organization

Section 2 deals with the adversarial setting, proving Theorems 1 and 2. Section 3 focuses on the random order setting, proving Theorem 3 and 4.

# 2 Adversarial Setting

In this section we consider the adversarial setting, and present both a tight randomized algorithm proving Theorem 1 and hardness for any deterministic algorithm (Theorem 2).

## 2.1 Tight Randomized Algorithm

In this section we consider the adversarial setting, and start by presenting our algorithm which appears in Algorithm 1. For simplicity of presentation, we assume the items are numbered according to the order the adversary chooses, *i.e.*, item 1 is the first to arrive, item 2 is the second to arrive and so forth. Hence, the algorithm performs $m$ iterations where in iteration $i$ the algorithm chooses what to do with item $i$: with a probability of $1/2$ it assigns it to the bidder with highest marginal value, with a probability of $1/4$ is assigns it to the bidder with the second highest marginal value, and so forth as long as the marginal value is non-negative. With the remainder probability item $i$ is discarded and not assigned to any bidder. Therefore, if there are $\ell$ bidders with non-negative marginal with respect to item $i$, item $i$ is discarded with a probability of $2^{-\ell}$. In what follows we use the notation $f_j(i|S)$ to denote the marginal value of item $i$ with respect to bidder $j$ assuming bidder $j$ was already assigned a subset $S \subseteq \mathcal{N}$ of items, *i.e.*, $f_j(i|S) \triangleq f_j(S \cup \{i\}) - f_j(S)$.

**Algorithm 1** Online Adversarial Algorithm.

$\forall j = 1, \ldots, n : S_j^0 \leftarrow \emptyset$.
**for** $i = 1, ..., m$ **do**
  $\forall j = 1, \ldots, n$: $S_j^i \leftarrow S_j^{i-1}$.
  $\forall r = 1, \ldots, n$: let $j_r$ be the bidder with the $r^{\text{th}}$ highest marginal w.r.t item $i$:
    $f_{j_1}(i|S_{j_1}^{i-1}) \geq f_{j_2}(i|S_{j_2}^{i-1}) \geq \ldots \geq f_{j_n}(i|S_{j_n}^{i-1})$.
  let $j^i$ be a random bidder such that $\Pr[j^i = j_r] = 2^{-r}$.
  **if** $f_{j^i}(i|S_{j^i}^{i-1}) \geq 0$ **then**
    $S_{j^i}^i \leftarrow S_{j^i}^i \cup \{i\}$
  **end**
**end**
return $S_1^m, S_2^m, \ldots, S_n^m$.

Let $O$ denote an offline optimal allocation for the problem: $O = (O_1, \ldots, O_n)$, where $O_j$ is the collection of items assigned to bidder $j$ in the optimal solution. Formally, $O$ is defined as follows[2]:

$$\underset{(O_1,\ldots,O_n)}{\arg\max} \left\{ \sum_{j=1}^{n} f_j(O_j) : \forall j \neq r \quad O_j \cap O_r = \emptyset, \quad \forall j = 1, \ldots, n \quad O_j \subseteq \mathcal{N} \right\}.$$

Let $S^i$ be the allocation induced by the algorithm at the end of the $i^{\text{th}}$ iteration, *i.e.*, $S^i = (S_1^i, S_2^i, \ldots, S_n^i)$. For every bidder $j$, define $H_j^i$ as follows:

$$H_j^i \triangleq (O_j \cap \{1, \ldots, i\}) \cup S_j^i,$$

and let $H^i = (H_1^i, \ldots, H_n^i)$. It is important to note that $H^i$ is not necessarily a feasible solution, since an item might be assigned to up to two bidders. For simplicity of presentation we use the notation of $f(S^i)$ to denote the value of the allocation $S^i$, *i.e.*, $f(S^i) \triangleq \sum_{j=1}^n f_j(S_j^i)$. Similarly, we use $f(H^i)$ to denote $\sum_{j=1}^n f_j(H_j^i)$ (though $H^i$ is not an allocation since items might be assigned to more than a single bidder).

In order to analyze the algorithm we denote by $P^i$ the profit gained during the $i^{\text{th}}$ iteration: $P^i \triangleq f(S^i) - f(S^{i-1})$. Note that $P^1 + \ldots + P^m = f(S^m) - f(S^0)$, where $S^m$ is the output of Algorithm 1 and $S^0$ is the empty allocation that does not assign any item to any of the bidders. Finally, we define the sequence: $K^i \triangleq f(H^i) - f(H^{i-1})$. Note that $K^1 + \ldots + K^m = f(H^m) - f(S^0)$ (note that $H^0 = S^0$).

The analysis of the competitive ratio of Algorithm 1 is essentially based on a single observation, which for every iteration upper bounds $K^i$ as a function of the expected gained profit. This is summarized in Lemma 6.

▶ **Lemma 6.** *$\forall i = 1, \ldots, m$ the following holds:* $\mathbb{E}[K^i] \leq 2 \cdot \mathbb{E}[P^i]$.

We use Lemma 6 to prove Theorem 1. The proof is essentially by summation over all iterations of the algorithm.

[2] Note that formally speaking this is actually a set, but for notational convenience, in this paper, when we refer to something as equalling the arg max, we will always mean it equals an element of the arg max set.

**Proof of Theorem 1.** We prove that the competitive ratio of Algorithm 1 is $1/4$. Lemma 6, when applied for every iteration $i$, implies that: $\mathbb{E}\left[\sum_{i=1}^{m} K^i\right] \leq 2 \cdot \mathbb{E}\left[\sum_{i=1}^{m} P^i\right]$. Hence, it follows that: $\mathbb{E}\left[f(H^m) - f(S^0)\right] \leq 2 \cdot \mathbb{E}\left[f(S^m) - f(S^0)\right]$. In fact, since $f(S^0)$ is non-negative, we have that:

$$\mathbb{E}\left[f(H^m)\right] \leq 2 \cdot \mathbb{E}\left[f(S^m)\right].$$

For every bidder $j$ and item $i$ we have that the probability that item $i$ is assigned to bidder $j$ is at most $1/2$, *i.e.*, $\Pr[i \in S_j^m] \leq 1/2$. Therefore, using Lemma 5 applied to the submodular function $h_j$, where $h_j : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ and $h_j(S) \triangleq f_j(S \cup O_j)$ for every $S \subseteq \mathcal{N}$, we can conclude that: $\mathbb{E}[f_j(S_j^m \cup O_j)] = \mathbb{E}[h_j(S_j^m)] \geq (1/2) \cdot h_j(\emptyset) = (1/2) \cdot f_j(O_j)$. Therefore,

$$\mathbb{E}[f(H^m)] = \sum_{j=1}^{n} \mathbb{E}[f_j(H_j^m)] = \sum_{j=1}^{n} \mathbb{E}[f_j(S_j^m \cup O_j)] \geq \frac{1}{2} \sum_{j=1}^{n} f_j(O_j) = \frac{1}{2} \cdot f(O).$$

Combining the above, we conclude that: $(1/4) \cdot f(O) \leq \mathbb{E}[f(S^m)]$. ◀

All that remains is to prove Lemma 6.

**Proof of Lemma 6.** For proof simplicity, let us start by adding a dummy bidder whose utility is the zero function. Clearly, this does not change the algorithm's performance or the value of $f(O)$, but it lets us assume (without loss of generality) that: (1) for every item $i$ the optimal solution $O$ allocates item $i$ to some bidder $k$, *i.e.*, $i \in O_k$; and (2) for every item $i$ at least one bidder has a non-negative marginal value, *i.e.*, $f_j(i|S_j^{i-1}) \geq 0$ for some bidder $j$.

Fix an iteration $i = 1, \ldots, m$ and condition on any possible realization $R_{i-1}$ of the random choices of the algorithm in the first $i-1$ iterations. Thus, $S^{i-1}$, $H^{i-1}$, and $j_1$ up to $j_m$ are deterministic and fixed given this conditioning, whereas $j^i$ is a random variable.

Recall that (without loss of generality) item $i$ is assigned to bidder $k$ by the optimal solution and that $j_r$ denotes the bidder with the $r^{\text{th}}$ highest marginal value with respect to item $i$ given the elements previously assigned to the bidder. Let us assume that the first $\ell \geq 1$ bidders have a non-negative marginal value with respect to item $i$, *i.e.*, $f_{j_1}(i|S_{j_1}^{i-1}) \geq f_{j_2}(i|S_{j_2}^{i-1}) \geq \ldots \geq f_{j_\ell}(i|S_{j_\ell}^{i-1}) \geq 0$ and if $\ell < m$: $f_{j_{\ell+1}}(i|S_{j_{\ell+1}}^{i-1}) < 0$. Moreover, assume that bidder $k$ has the $t^{\text{th}}$ largest marginal, *i.e.*, $k = j_t$. Let us now bound the expected change from $f(H^{i-1})$ to $\mathbb{E}[f(H^i)|R_{i-1}]$, *i.e.*, $\mathbb{E}[K^i|R_{i-1}]$, given the assumption that $t \leq \ell$, *i.e.*, bidder $k$ to which the optimal solution assigned item $i$ is among the $\ell$ bidders who have a non-negative marginal value:

$$\begin{aligned}
&\mathbb{E}[f(H^i)|R_{i-1}] - f(H^{i-1}) \\
=&f_k(S_k^{i-1} \cup (O_k \cap \{1, \ldots, i-1\}) \cup \{i\}) - f_k(S_k^{i-1} \cup (O_k \cap \{1, \ldots, i-1\})) + \qquad (1) \\
&\sum_{r=1}^{\ell} \frac{\mathbf{1}_{\{j_r \neq k\}}}{2^r} \left\{ f_{j_r}((O_{j_r} \cap \{1, \ldots, i-1\}) \cup S_{j_r}^{i-1} \cup \{i\}) - f_{j_r}((O_{j_r} \cap \{1, \ldots, i-1\}) \cup S_{j_r}^{i-1}) \right\}.
\end{aligned}$$

The equality in (1) follows from the definitions of $H^i$ and the algorithm, as well as the fact that $i \in O_k$. We note that that (1) can be upper bounded as follows:

$$\leq f_k(i|S_k^{i-1}) + \sum_{r=1}^{\ell} \frac{\mathbf{1}_{\{j_r \neq k\}} f_{j_r}(i|S_{j_r}^{i-1})}{2^r}. \qquad (2)$$

The inequality in (2) follows from the decreasing marginals property of the submodular utilities. Next, let us rewrite (2):

$$= f_{j_t}(i|S_{j_t}^{i-1}) \cdot \left(1 - \frac{1}{2^t}\right) + \sum_{r=1}^{\ell} \frac{f_{j_r}(i|S_{j_r}^{i-1})}{2^r} \tag{3}$$

$$= f_{j_t}(i|S_{j_t}^{i-1}) \cdot \left(\sum_{r=1}^{t} 2^{-r}\right) + \sum_{r=1}^{\ell} \frac{f_{j_r}(i|S_{j_r}^{i-1})}{2^r}. \tag{4}$$

The equality in (3) holds since bidder $k$ has the $t^{\text{th}}$ largest marginal, *i.e.*, $k = j_t$, and $t \leq \ell$. Also, the equality in (4) follows from the value of a geometric sum. Moreover, we note that (4) can be further upper bounded:

$$\leq \sum_{r=1}^{t} \frac{f_{j_r}(i|S_{j_r}^{i-1})}{2^r} + \sum_{r=1}^{\ell} \frac{f_{j_r}(i|S_{j_r}^{i-1})}{2^r}. \tag{5}$$

In the above, the inequality in (5) is true since bidder $j_r$ has the $r^{\text{th}}$ largest marginal value, *i.e.*, for every $r \leq t$: $f_{j_r}(i|S_{j_r}^{i-1})) \geq f_{j_t}(i|S_{j_t}^{i-1})$. Next, we upper bound (5) as follows:

$$\leq 2 \cdot \sum_{r=1}^{\ell} \frac{f_{j_r}(i|S_{j_r}^{i-1})}{2^r}. \tag{6}$$

We note that the inequality in (6) follows since we assumed $t \leq \ell$, *i.e.*, bidder $k$ is among the $\ell$ bidders with non-negative marginal values. Hence, the first sum in (5) can be extended to include all $\ell$ bidders with non-negative marginal value. Finally, we note that (6) equals the following by the definition of Algorithm 1:

$$= 2 \cdot \mathbb{E}[P^i|R_{i-1}].$$

Hence, we can conclude that $\mathbb{E}[K^i|R_{i-1}] \leq 2 \cdot \mathbb{E}[P^i|R_{i-1}]$ as desired.

We note that if $t > \ell$ then the latter inequality trivially holds (the above proof works until (2) in which the first term is negative and thus can be dropped which implies that $\mathbb{E}[K^i|R_{i-1}] \leq \mathbb{E}[P^i|R_{i-1}]$ and hence that $\mathbb{E}[K^i|R_{i-1}] \leq 2 \cdot \mathbb{E}[P^i|R_{i-1}]$, since $P_i$ is non-negative). Thus, using the law of total expectation over all possible outcomes $R_{i-1}$ the proof is complete. ◀

## 2.2 Deterministic Hardness

**Proof of Theorem 2.** We present an instance for which any online deterministic algorithm cannot achieve a competitive ratio better than $1/M$ for every $M > 0$ versus an adversary. We consider an instance with a single bidder $B = \{b\}$ and two items $\mathcal{N} = \{v_1, v_2\}$. The items arrive according to their index in an online manner, *i.e.*, item $v_1$ is the first to arrive, and item $v_2$ is the second to arrive. Once item $v_1$ arrives, any deterministic algorithm can query $f$ on subsets of elements that can contain only $v_1$. Hence, we only need to define $f$ on $\emptyset$ and $\{v_1\}$. We define $f$ as follows: $f(\emptyset) = 0, f(\{v_1\}) = 1$. How this utility function is extended to a submodular function over all subsets of items depends on what the algorithm chooses to do once item $v_1$ arrives.

The first case is when the algorithm chooses not to assign $v_1$ to $b$. In this case we extend the above definition of $f$ by setting the contribution of $v_2$ to be linearly zero, *i.e,*, $f(\{v_2\}) \triangleq 0$ and $f(\{v_1, v_2\}) \triangleq 1$. One can verify that the resulting utility function $f$ is indeed submodular.

In this case, the value of an optimal solution to the instance equals 1, since item $v_1$ can be assigned to bidder $b$ by the optimal solution. However, every deterministic algorithm that does not assign $v_1$ to $b$ has a value of 0. Thus, we conclude a competitive ratio of 0 in this case.

The second case is when the algorithm chooses to assign $v_1$ to bidder $b$. In this case we extend the above definition of $f$ as follows: $f(\{v_2\}) \triangleq M$ and $f(\{v_1, v_2\}) \triangleq 0$. One can verify that the resulting utility function $f$ is indeed submodular. In this case, the value of an optimal solution to the instance equals $M$, since the optimal solution can choose to assign only $\{v_2\}$ to $b$. However, every deterministic algorithm that assigns $v_1$ to bidder $b$ achieves a value of at most 1. Thus, we conclude a competitive ratio of at most $1/M$ in this case.

In conclusion, for this instance, no deterministic algorithm can achieve any competitive ratio better than $1/M$ in the online adversarial setting, for every constant $M$. ◀

# 3 Uniform Random Order Setting

In this section we focus on the uniform random order setting. Recall that it is known that if the RRG algorithm provides an approximation of $\alpha$ for maximizing a general submodular function given a partition matroid independence constraint, then it also provides an online algorithm in the random order setting which achieves a competitive ratio of $\alpha$ (see brief discussion in Section 1).

To simplify the presentation of our improved analysis, we present a "smooth" version of the RRG algorithm of [10]. Though the analysis of the smoothed version is similar to the original RRG (assuming enough iterations are performed), we believe it is easier to analyze since all iterations have the same probabilistic distribution (whereas in the original RRG this is not the case). For simplicity of presentation, we focus on the case the matroid is a partition matroid (for a general matroid we defer to a full version of the paper). Recall that this special case already captures SW.

## 3.1 Partition Matroid

We are given a partition matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ over a ground set $\mathcal{N}$ which is partitioned into disjoint non-empty sets $P_1, P_2, \ldots, P_k$. The goal is to choose a subset $S \subseteq \mathcal{I}$, *i.e.*, $S$ contains at most one element from each set $P_j$, that maximizes a given non-negative (general) submodular function $f$.

We call our algorithm smooth since the random choices of the algorithm are always uniform, no matter how many iterations the algorithm performed so far. Specifically, the smooth algorithm chooses in every iteration a part uniformly at random from *all* $k$ parts of $\mathcal{N}$, and adds the best element in the chosen part assuming that part does not intersect what the algorithm chose so far. The original RRG chooses a part uniformly at random from parts that do not intersect what the algorithm chose so far, and adds the best element in the chosen part. Thus, the number of iterations the smooth algorithm can perform is unlimited. A formal description of the algorithm for partition matroids is given as Algorithm 2, and we note that the number of iterations $T$ is a parameter given to the algorithm. For a comparison of Algorithm 2 and the original RRG we defer to to a full version of the paper.

Given any $S \subseteq \mathcal{N}$, we denote the parts of the partition of $\mathcal{N}$ that do not intersect $S$ by $I(S)$ ,*i.e.*, $I(S) \triangleq \{j = 1, ..., k | P_j \cap S = \emptyset\}$. Without loss of generality, we assume every part $P_j$ is padded with a dummy element (a different dummy element for every $P_j$) that linearly contributes zero to the objective $f$. Hence, without loss of generality, $|OPT| = k$, where we denote by $OPT$ an optimal solution to the problem: $OPT \triangleq \arg\max\{f(S) | S \in \mathcal{I}\}$.

**Algorithm 2** Smooth Residual Random Greedy (Partition Matroid).

$S_0 \leftarrow \emptyset$.
**for** $i = 1, ..., T$ **do**
  $S_i \leftarrow S_{i-1}$.
  $M_i \leftarrow \bigcup_{j \in I(S_{i-1})} \{\arg\max \{f(S_{i-1} \cup \{u\}) - f(S_{i-1}) | u \in P_j\}\}$.
  Let $j$ be a uniformly random number from $\{1, ..., k\}$.
  **if** $j \in I(S_{i-1})$ **then**
    Let $u_i$ be the element from $P_j$ in $M_i$ and $S_i \leftarrow S_i \cup \{u_i\}$.
  **end**
**end**
Return $S_T$.

For every set $S \in \mathcal{I}$ we denote $O_S \triangleq \arg\max_A \{f(S \cup A) | S \cup A \in \mathcal{I}, A \subseteq \mathcal{N} \setminus S\}$, *i.e.*, $O_S$ is the best extension of $S$ to an independent set. Recalling that every part $P_j$ is padded with a dummy element that linearly contributes zero to the objective implies that $|O_S| = k - |S|$. One should note that $O_\emptyset = OPT$, and thus $O_{S_0} = OPT$. Our analysis tracks $f(O_{S_i} \cup S_i)$ as $S_i$ changes throughout the algorithm. Intuitively, $f(O_{S_i} \cup S_i)$ deteriorate as more elements are added to $S_i$. Building on the above intuition, the following two lemmas establish a system of joint recursive formulas for $\mathbb{E}[f(S_i)]$ and $\mathbb{E}[f(O_{S_i} \cup S_i)]$.

▶ **Lemma 7.** *For every $i = 1, \ldots, T$:*

$$\mathbb{E}[f(S_i)] - \mathbb{E}[f(S_{i-1})] \geq \frac{1}{k} \cdot \mathbb{E}\left[f\left(O_{S_{i-1}} \cup S_{i-1}\right) - f(S_{i-1})\right].$$

**Proof.** Fix $i = 1, \ldots, T$ and condition on any possible realization of the choices of the algorithm in the first $i-1$ iterations. Thus, $S_{i-1}$, $M_i$, and $O_{S_{i-1}}$ are deterministic and fixed given this conditioning, and $u_i$ and $S_i$ are the only random variables. For the remainder of the proof all the probabilities and expectations are conditioned on this possible realization.

$$\mathbb{E}[f(S_i)] - f(S_{i-1}) = \frac{1}{k} \sum_{u \in M_i} \{f(S_{i-1} \cup \{u\}) - f(S_{i-1})\} \tag{7}$$

$$\geq \frac{1}{k} \sum_{u \in O_{S_{i-1}}} \{f(S_{i-1} \cup \{u\}) - f(S_{i-1})\} \tag{8}$$

$$\geq \frac{1}{k} \left\{f(S_{i-1} \cup O_{S_{i-1}}) - f(S_{i-1})\right\} \tag{9}$$

In the above, the equality in (7) follows from the algorithm's definition. The inequality in (8) follows from the greedy choice of $M_i$, and the inequality in (9) follows from the submodularity of $f$. We conclude the proof by unfixing the conditioning and taking an expectation over all possible such events (the law of total expectation). ◀

The following lemma provides a recursive formula for $\mathbb{E}[f(O_{S_i} \cup S_i)]$ whose novelty is in the added contribution of $f(S_{i-1})$. Without the added $\mathbb{E}[f(S_{i-1})]/k$ term the resulting approximation will be $1/4$ as in [10].

▶ **Lemma 8.** *For every $i = 1, \ldots, T$:*

$$\mathbb{E}[f(O_{S_i} \cup S_i)] \geq \left(1 - \frac{2}{k}\right) \cdot \mathbb{E}\left[f\left(O_{S_{i-1}} \cup S_{i-1}\right)\right] + \frac{1}{k} \cdot \mathbb{E}[f(S_{i-1})].$$

**Proof.** Fix $i = 1, \ldots, T$ and condition on any possible realization of the choices of the algorithm in the first $i-1$ iterations. Thus, $S_{i-1}$, $M_i$, and $O_{S_{i-1}}$ are deterministic and fixed given this conditioning, and $u_i$, $S_i$, and $O_{S_i}$ are the only random variables. For the remainder of the proof all the probabilities and expectations are conditioned on this possible realization.

For every element $u \in M_i$ we denote by $P(u)$ the index of the part of the partition of $\mathcal{N}$ which $u$ belongs to. Formally, $P(u) = j$ if and only if $u \in P_j$. We define $h : M_i \to O_{S_{i-1}}$ to be a bijection mapping every element $u \in M_i$ to an element of $O_{S_{i-1}}$ in such a way that $P(u) = P(h(u))$. One should note that our assumption that every $P_j$ contains a dummy element that contributes zero to the objective implies that $I(S_{i-1}) = I(O_{S_{i-1}})$, and hence $h$ is well defined. Thus,

$$\begin{aligned}
&\mathbb{E}\left[f\left(O_{S_i} \cup S_i\right)\right] - f\left(O_{S_{i-1}} \cup S_{i-1}\right) \\
=&\frac{1}{k}\sum_{u \in M_i}\left\{f\left(S_{i-1} \cup \{u\} \cup O_{S_{i-1}\cup\{u\}}\right) - f\left(O_{S_{i-1}} \cup S_{i-1}\right)\right\} && (10)\\
\geq&\frac{1}{k}\sum_{u \in M_i}\left\{f\left(S_{i-1} \cup \{u\} \cup \left(O_{S_{i-1}} \setminus \{h(u)\}\right)\right) - f\left(O_{S_{i-1}} \cup S_{i-1}\right)\right\}. && (11)
\end{aligned}$$

In the above, the equality in (10) follows from the algorithm's definition. The inequality in (11) follows from the observation that $O_{S_{i-1}\cup\{u\}}$ is the best extension of $S_{i-1} \cup \{u\}$ whereas $O_{S_{i-1}} \setminus \{h(u)\}$ is just an extension of $S_{i-1} \cup \{u\}$, *i.e.*,

$$f(S_{i-1} \cup \{u\} \cup O_{S_{i-1}\cup\{u\}}) \geq f(S_{i-1} \cup \{u\} \cup (O_{S_{i-1}} \setminus \{h(u)\})).$$

We note that (11) equals the following:

$$=\frac{1}{k}\sum_{u \in M_i | u \neq h(u)}\left\{f\left(S_{i-1} \cup \{u\} \cup \left(O_{S_{i-1}} \setminus \{h(u)\}\right)\right) - f\left(O_{S_{i-1}} \cup S_{i-1}\right)\right\}. \quad (12)$$

The equality in (12) holds since if $u = h(u)$ then $\{u\} \cup (O_{S_{i-1}} \setminus \{h(u)\}) = O_{S_{i-1}}$, and therefore summation can be reduced to all candidate elements $u \in M_i$ satisfying $u \neq h(u)$. Moreover, we note that (12) can be lower bounded as follows:

$$\begin{aligned}
\geq&\frac{1}{k}\sum_{u \in M_i | u \neq h(u)}\left\{f\left(S_{i-1} \cup \{u\} \cup O_{S_{i-1}}\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\right\}+ && (13)\\
&\frac{1}{k}\sum_{u \in M_i | u \neq h(u)}\left\{f\left(S_{i-1} \cup \left(O_{S_{i-1}} \setminus \{h(u)\}\right)\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\right\}.
\end{aligned}$$

The inequality in (13) follows from submodularity since summation is restricted only to candidates $u \in M_i$ satisfying $u \neq h(u)$ and hence: $f(S_{i-1} \cup \{u\} \cup (O_{S_{i-1}} \setminus \{h(u)\})) + f(S_{i-1} \cup O_{S_i-1}) \geq f(S_{i-1} \cup \{u\} \cup O_{S_{i-1}}) + f(S_{i-1} \cup (O_{S_{i-1}} \setminus \{h(u)\}))$.

We further lower bound (13) in the following way:

$$\begin{aligned}
\geq&\frac{1}{k}\sum_{u \in M_i}\left\{f\left(S_{i-1} \cup \{u\} \cup O_{S_{i-1}}\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\right\}+ && (14)\\
&\frac{1}{k}\sum_{u \in M_i}\left\{f\left(S_{i-1} \cup \left(O_{S_{i-1}} \setminus \{h(u)\}\right)\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\right\}.
\end{aligned}$$

When examining the inequality in (14) let us start with the first sum. We note that if $u = h(u)$ for some $u \in M_i$ then $u \in O_{S_{i-1}}$, *i.e.*, $\{u\} \cup O_{S_{i-1}} = O_{S_{i-1}}$. Thus, extending the first sum to all $u \in M_i$ (regardless of whether $u$ equals $h(u)$ or not) does not change the

first sum. Focusing on the second sum, we note that for every $u \in M_i$, regardless of whether $u$ equals $h(u)$ or not, the following holds: $f(S_{i-1} \cup (O_{S_{i-1}} \setminus \{h(u)\})) \leq f(S_{i-1} \cup O_{S_{i-1}})$. The reason for the latter is that $O_{S_{i-1}}$ is the best extension of $S_{i-1}$ whereas $O_{S_{i-1}} \setminus \{h(u)\}$ is some extension of $S_{i-1}$. Hence, adding to the second sum all terms corresponding to $u \in M_i$, where $u = h(u)$, can only decrease the second sum. Therefore, we conclude that the inequality in (14) holds. Finally, we lower bound (14) as follows:

$$\geq \frac{1}{k}\{f\left(S_{i-1} \cup O_{S_{i-1}} \cup M_i\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\} + \tag{15}$$
$$\frac{1}{k}\{f\left(S_{i-1} \cup O_{S_{i-1}} \setminus O_{S_{i-1}}\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\}$$
$$\geq \frac{1}{k}\{f(S_{i-1}) - 2 \cdot \left(f\left(S_{i-1} \cup O_{S_{i-1}}\right)\right)\} \tag{16}$$

The inequality in (15) follows from submodularity which implies the following two:

$$\sum_{u \in M_i} \{f\left(S_{i-1} \cup \{u\} \cup O_{S_{i-1}}\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\} \geq f\left(S_{i-1} \cup O_{S_{i-1}} \cup M_i\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)$$

$$\sum_{u \in M_i} \{f\left(S_{i-1} \cup O_{S_{i-1}} \setminus \{h(u)\}\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right)\} \geq f\left(S_{i-1} \cup O_{S_{i-1}} \setminus O_{S_{i-1}}\right) - f\left(S_{i-1} \cup O_{S_{i-1}}\right).$$

We note that the inequality in (16) follows from the non-negativity of $f$ which implies that: $f\left(S_{i-1} \cup O_{S_{i-1}} \cup M_i\right) \geq 0$ and the fact that $S_{i-1} \cup O_{S_{i-1}} \setminus O_{S_{i-1}} = S_{i-1}$.

We conclude the proof by unfixing the conditioning and taking an expectation over all possible such events (the law of total expectation). ◀

The following lemma lower bounds the solution to the system of joint recursive formulas presented in Lemmas 7 and 8 (its proof is deferred to a full version of the paper). For simplicity of presentation we introduce two absolute constants: $a \triangleq (3 - \sqrt{5})/2 \approx 0.381966$ and $b \triangleq (3 + \sqrt{5})/2 \approx 2.61803$.

▶ **Lemma 9.** *For every $i = 0, 1, \ldots, T$ the following hold:*

$$\mathbb{E}[f(S_i)] \geq \frac{f(OPT)}{\sqrt{5}} \left(\left(1 - \frac{a}{k}\right)^i - \left(1 - \frac{b}{k}\right)^i\right) \tag{17}$$

$$\mathbb{E}\left[f(O_{S_i} \cup S_i)\right] \geq \frac{f(OPT)}{2\sqrt{5}} \left((\sqrt{5} - 1)\left(1 - \frac{a}{k}\right)^i + (\sqrt{5} + 1)\left(1 - \frac{b}{k}\right)^i\right). \tag{18}$$

The following lemma establishes the approximation guarantee of Algorithm 2.

▶ **Lemma 10.** *Algorithm 2 achieves an approximation ratio of at least* 0.27493 *for the problem of maximizing a non-monotone submodular function subject to a partition matroid independence constraint.*

**Proof.** By our assumption of the existence of dummy elements, no element of $M_i$ has a negative marginal value, in any iteration $i$. We get that always for every $i$, $f(S_i) \geq f(S_{i-1})$ (note that this inequality holds for the random variables $S_i$ and $S_{i-1}$). Therefore, it suffices to show that $\mathbb{E}[f(S_i)] \geq 0.274 \cdot f(OPT)$ for some $i = 1, \ldots, T$.

Observe that setting $i = x^*k$, where $x^* = \ln(b/a)/\sqrt{5} = \ln\left((3 + \sqrt{5})/(3 - \sqrt{5})\right)/\sqrt{5} \approx 0.86$, alongside Lemma 9, implies that:

$$\mathbb{E}[f(S_{x^*k})] \geq \frac{f(OPT)}{\sqrt{5}} \left(\left(1 - \frac{a}{k}\right)^{x^*k} - \left(1 - \frac{b}{k}\right)^{x^*k}\right) \geq \frac{f(OPT)}{\sqrt{5}} \left(e^{-x^*a} - e^{-x^*b}\right). \tag{19}$$

The inequality above follows from the observation that $(1-a/k)^{x^*k} - (1-b/k)^{x^*k}$, as a function of $k$, is monotone decreasing for every $k \geq 3$ (the proof of this technical observation is deferred to a full version of the paper). Thus, the inequality follows by taking the limit $k \to \infty$. The lemma follows by plugging in the above values of $a$, $b$, and $x^*$. ◀

**Proof of Theorem 3.** Follows immediately from Lemma 10. ◀

## References


1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. *Online Vertex-Weighted Bipartite Matching and Single-bid Budgeted Allocations*, pages 1253–1264. SIAM, 2011. doi:10.1137/1.9781611973082.95.

2 Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Operations Research*, 62(4):876–890, 2014.

3 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 434–443, USA, 2007. Society for Industrial and Applied Mathematics.

4 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1497–1514, 2014.

5 Siddharth Barman, Seeun Umboh, Shuchi Chawla, and David Malec. Secretary problems with convex costs. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 75–87, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

6 Mohammadhossein Bateni, Mohammadtaghi Hajiaghayi, and Morteza Zadimoghaddam. Submodular secretary problem and extensions. *ACM Trans. Algorithms*, 9(4), October 2013. doi:10.1145/2500121.

7 N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 44(5):1384–1402, 2015.

8 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.

9 Niv Buchbinder, Moran Feldman, Yuval Filmus, and Mohit Garg. Online submodular maximization: Beating 1/2 made simple. *Mathematical Programming*, 183(1-2):149–169, 2020.

10 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1433–1452, 2014.

11 Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. *ACM Trans. Algorithms*, 15(3), June 2019. doi:10.1145/3309764.

12 Niv Buchbinder, Kamal Jain, and Joseph (Seffi) Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms – ESA 2007*, pages 253–264, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

13 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, pages 182–196, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

14 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

15 Deeparnab Chakrabarty and Gagan Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap. *SIAM Journal on Computing*, 39(6):2189–2211, 2010. doi:10.1137/080735503.

16 Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: Online keyword matching with budgeted bidders under random permutations. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, EC '09, pages 71–78, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1566374.1566384.

17 Nikhil R Devanur, Zhiyi Huang, Nitish Korula, Vahab S Mirrokni, and Qiqi Yan. Whole-page optimization and submodular welfare maximization with online bidders. *ACM Transactions on Economics and Computation (TEAC)*, 4(3):1–20, 2016.

18 Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. *Mathematics of Operations Research*, 35(1):1–13, 2010. doi:10.1287/moor.1090.0436.

19 Shahar Dobzinski and Michael Schapira. An improved approximation algorithm for combinatorial auctions with submodular bidders. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1064–1073, USA, 2006. Society for Industrial and Applied Mathematics.

20 E. B. Dynkin. The optimum choice of the instant for stopping a markov process, 1963.

21 Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 248–257, 2016. doi:10.1109/FOCS.2016.34.

22 Alina Ene and Huy L. Nguyen. A nearly-linear time algorithm for submodular maximization with knapsack, partition and graphical matroid constraints. *CoRR*, abs/1709.09767, 2018.

23 Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing*, 39(1):122–142, 2009. doi:10.1137/070680977.

24 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

25 Uriel Feige and Jan Vondrak. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 667–676, 2006. doi:10.1109/FOCS.2006.14.

26 Uriel Feige and Jan Vondrák. The submodular welfare problem with demand queries. *Theory of Computing*, 6(11):247–290, 2010. doi:10.4086/toc.2010.v006a011.

27 Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S Mirrokni, and Cliff Stein. Online stochastic packing applied to display ad allocation. In *European Symposium on Algorithms*, pages 182–194. Springer, 2010.

28 Jon Feldman, Nitish Korula, Vahab Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In Stefano Leonardi, editor, *Internet and Network Economics*, pages 374–385, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

29 Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 117–126, 2009. doi:10.1109/FOCS.2009.72.

30 Moran Feldman. Maximizing symmetric submodular functions. *ACM Transactions on Algorithms (TALG)*, 13(3):1–36, 2017.

31 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 570–579, 2011. doi:10.1109/FOCS.2011.46.

32 Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Nonmonotone submodular maximization via a structural continuous greedy algorithm. In *ICALP*, pages 342–353, 2011.

33 Moran Feldman, Ola Svensson, and Rico Zenklusen. *A Simple O(log log(rank))-Competitive Algorithm for the Matroid Secretary Problem*, pages 1189–1201. SIAM, 2015. doi:10.1137/1.9781611973730.79.

34 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. SIAM, 2011.

**35** Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, volume 8, pages 982–991. Citeseer, 2008.

**36** Corinna Gottschalk and Britta Peis. Submodular function maximization on the bounded integer lattice. In *Approximation and Online Algorithms*. Springer International Publishing, 2015.

**37** Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In Amin Saberi, editor, *Internet and Network Economics*, pages 246–257, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

**38** Christopher Harshaw, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. The power of subsampling in submodular maximization. *Mathematics of Operations Research*, 47(2):1365–1393, 2022.

**39** Xinran He and Yan Liu. Not enough data? joint inferring multiple diffusion networks via network generation priors. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 465–474, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3018661.3018675.

**40** Bala Kalyanasundaram and Kirk R. Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1):319–325, 2000. doi:10.1016/S0304-3975(99)00140-1.

**41** Mikhail Kapralov, Ian Post, and Jan Vondrák. Online and stochastic variants of welfare maximization. *CoRR*, abs/1204.1025, 2012. arXiv:1204.1025.

**42** R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 352–358, New York, NY, USA, 1990. Association for Computing Machinery. doi:10.1145/100216.100262.

**43** Subhash Khot, Richard J. Lipton, Evangelos Markakis, and Aranyak Mehta. Inapproximability results for combinatorial auctions with submodular utility functions. In *Proceedings of the First International Conference on Internet and Network Economics*, WINE'05, pages 92–101, 2005.

**44** Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. doi:10.1016/S0020-0190(99)00031-9.

**45** Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 630–631, USA, 2005. Society for Industrial and Applied Mathematics.

**46** Nitish Korula, Vahab Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018. doi:10.1137/15M1051142.

**47** Andreas Krause and Carlos Guestrin. A note on the budgeted maximization of submodular functions. *Technical Report CMU-CALD-05-103*, 2005.

**48** Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Math. Oper. Res.*, 38(4):729–739, November 2013.

**49** Oded Lachish. O(log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 326–335, 2014. doi:10.1109/FOCS.2014.42.

**50** Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics*, 23(4):2053–2078, 2010.

**51** Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, November 2010.

52 Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, EC '01, pages 18–28, 2001.

53 Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, 2010.

54 Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, pages 510–520, 2011.

55 Shengzhong Liu, Zhenzhe Zheng, Fan Wu, Shaojie Tang, and Guihai Chen. Context-aware data quality estimation in mobile crowdsensing. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017. doi:10.1109/INFOCOM.2017.8057033.

56 Tengyu Ma, Bo Tang, and Yajun Wang. The simulated greedy algorithm for several submodular matroid secretary problems. *Theory of Computing Systems*, 58(4):681–706, May 2016. doi:10.1007/s00224-015-9642-4.

57 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, STOC '11, pages 597–606, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1993636.1993716.

58 Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22–es, October 2007. doi:10.1145/1284320.1284321.

59 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

60 G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978. doi:10.1287/moor.3.3.177.

61 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions–i. *Math. Program.*, 14(1):265–294, December 1978. doi:10.1007/BF01588971.

62 Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting prices. *Journal of Economic Theory*, 129(1):192–224, 2006. doi:10.1016/j.jet.2004.10.007.

63 Xinghao Pan, Stefanie Jegelka, Joseph E Gonzalez, Joseph K Bradley, and Michael I Jordan. Parallel double greedy submodular maximization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 118–126. Curran Associates, Inc., 2014.

64 Ramakumar Pasumarthi, Ramasuri Narayanam, and Balaraman Ravindran. Near optimal strategies for targeted marketing in social networks. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, pages 1679–1680, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

65 Paulo Shakarian, Joseph Salmento, William Pulleyblank, and John Bertetto. Reducing gang violence through network influence based targeting of social programs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 1829–1836, 2014.

66 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, January 2004.

**67** Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. Optimal online assignment with forecasts. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 109–118, 2010.

**68** J. Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

**69** Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 67–74, 2008.

**70** Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 783–792, 2011.

# First Order Logic and Twin-Width in Tournaments

**Colin Geniet**
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

**Stéphan Thomassé**
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

## Abstract

We characterise the classes of tournaments with tractable first-order model checking. For every hereditary class of tournaments $\mathcal{T}$, first-order model checking either is fixed parameter tractable, or is AW[$*$]-hard. This dichotomy coincides with the fact that $\mathcal{T}$ has either bounded or unbounded twin-width, and that the growth of $\mathcal{T}$ is either at most exponential or at least factorial. From the model-theoretic point of view, we show that NIP classes of tournaments coincide with bounded twin-width. Twin-width is also characterised by three infinite families of obstructions: $\mathcal{T}$ has bounded twin-width if and only if it excludes at least one tournament from each family. This generalises results of Bonnet et al. on ordered graphs.

The key for these results is a polynomial time algorithm which takes as input a tournament $T$ and computes a linear order $<$ on $V(T)$ such that the twin-width of the birelation $(T, <)$ is at most some function of the twin-width of $T$. Since approximating twin-width can be done in FPT time for an ordered structure $(T, <)$, this provides a FPT approximation of twin-width for tournaments.

**2012 ACM Subject Classification** Theory of computation → Finite Model Theory; Mathematics of computing → Graph algorithms; Mathematics of computing → Graph enumeration

**Keywords and phrases** Tournaments, twin-width, first-order logic, model checking, NIP, small classes

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.53

**Related Version** *Full Version*: `https://arxiv.org/abs/2207.07683` [16]

**Funding** The authors were partially supported by the ANR projects TWIN-WIDTH (ANR-21-CE48-0014-01) and DIGRAPHS (ANR-19-CE48-0013-01).

**Acknowledgements** The authors would like to thank Édouard Bonnet for stimulating discussions on this topics, and Szymon Toruńczyk for helpful explanations on the notion of restrained classes.

## 1 Introduction

Tournaments can represent the outcome of a ranking of candidates, which need not be a total order. E.g., in the Condorcet voting paradox, three referees whose preference lists are $(A, B, C)$, $(B, C, A)$, and $(C, A, B)$, lead to a cycle $A \leftarrow B \leftarrow C \leftarrow A$ in the preference relation. Classical algorithmic problems arise from trying to choose a subset of winners: the Dominating Set (DS) problem asks for a subset $D$ which is preferred to any other candidate, i.e. for any $y \notin D$, there is some $x \in D$ which is preferred to $y$; and the Feedback Vertex Set (FVS) problem asks to build a preference order by ignoring a subset of candidates.

These problems can be parameterized by the size $k$ of the desired solution. A problem is *fixed parameter tractable* (FPT) if it admits an algorithm running in time $O(f(k) \cdot n^c)$, for some function $f$ and constant $c$. It is known that FVS is FPT for tournaments [18], whereas DS is unlikely to be FPT. However general tournaments may not be representative of usual instances: for example, majority voting tournaments with a fixed number $r$ of referees form a very restricted class. A cornerstone paper by Alon et al. [2], based on Vapnik-Chervonenkis dimension, shows that $k$-DS is trivially FPT on $r$-majority tournaments, because the size of a minimum dominating set is bounded by $f(r)$. This exemplifies how difficult problems can become easy on restricted classes, here bounded VC-dimension.


To put these questions in a much broader perspective, remark that the previous problems can be expressed in first-order logic (FO). A $k$-DS is described by the formula

$$\exists x_1, x_2, \dots, x_k. \forall y.\ (y = x_1) \vee (y \to x_1) \vee \dots \vee (y = x_k) \vee (y \to x_k).$$

That $k$-FVS is also expressible in first-order logic is only true in tournaments, and not in general graphs. It is based on the simple remark that a tournament is acyclic if and only if it is transitive, i.e. it has no directed 3-cycle, which is easily expressed in FO. Thus $k$-DS and $k$-FVS are instances of the FO Model Checking (or FOMC) problem: given as input a tournament $T$ and a first-order formula $\phi$, does $T$ satisfies $\phi$? FO model checking is difficult on the class of all graphs [12], and using back-and-forth FO encodings, one can show that it is just as hard on tournaments. We investigate which subclasses of tournaments admit an FPT algorithm for FO model checking.

## 1.1 Main results

We prove a dichotomy: in any class $\mathcal{T}$ of tournaments (closed under subtournaments), FOMC is either FPT or AW[$*$]-complete. The key of this dichotomy is *twin-width* (tww), a complexity parameter introduced by Bonnet et al. [7]: FOMC in $\mathcal{T}$ is FPT if $\mathcal{T}$ has bounded twin-width, and AW[$*$]-complete otherwise. This dichotomy coincides with a model theoretic characterisation: the class $\mathcal{T}$ has bounded twin-width if and only if it is NIP, meaning that arbitrary graphs cannot be described from tournaments in $\mathcal{T}$ through a fixed FO formula. This equivalence of twin-width and NIP, called *delineation*, was conjectured for tournaments in [4]. The equivalence between NIP and FPT FO model checking also confirms the *nowhere FO dense* conjecture of Gajarský et al. [15] for tournaments.

Furthermore, the dichotomy for FO model checking coincides with a gap in the growth function of the class $\mathcal{T}$, i.e. the number of tournaments of $\mathcal{T}$ on $n$ vertices up to isomorphism. If $\mathcal{T}$ has bounded twin-width, then its growth is at most $2^{O(n)}$, whereas it is at least $(\lfloor n/2 \rfloor - 1)!$ when twin-width is unbounded. This exponential/factorial gap generalises the Marcus-Tardos theorem on permutations avoiding a fixed pattern [19]. It may also be compared to results of Boudabbous and Pouzet [9] which show that hereditary classes of tournaments have growth either at most polynomial or at least exponential.

▶ **Theorem 1.1.** *Let $\mathcal{T}$ be a hereditary class of tournaments. Under the assumption FPT $\neq$ AW[$*$], the following are equivalent:*

1. *$\mathcal{T}$ has bounded twin-width,*
2. *FO model checking in $\mathcal{T}$ is FPT,*
3. *FO model checking in $\mathcal{T}$ is not AW[$*$]-complete,*
4. *$\mathcal{T}$ does not FO interpret the class of all graphs,*
5. *$\mathcal{T}$ is monadically NIP, i.e. does not FO transduce all graphs,*
6. *the growth of $\mathcal{T}$ is at most $c^n$ for some constant $c$,*
7. *the growth of $\mathcal{T}$ is less than $(\lfloor \frac{n}{2} \rfloor - 1)!$.*

These equivalences are completed by three minimal classes of obstructions, characterising twin-width by excluded substructures. These obstructions encode arbitrary permutations.

▶ **Theorem 1.2.** *There are three hereditary classes $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$ such that a hereditary class $\mathcal{T}$ of tournaments has unbounded twin-width if and only if one of $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$ is a subclass of $\mathcal{T}$.*

Finally, we show that there is a fixed parameter tractable algorithm which approximates twin-width of tournaments up to some function.

▶ **Theorem 1.3.** *There is a function $f : \mathbb{N} \to \mathbb{N}$ and an FPT algorithm, which given as input a tournament $T$, produces either a witness* $\mathrm{tww}(T) \leqslant f(k)$*, or a witness that* $\mathrm{tww}(T) \geqslant k$*.*

These results can be generalised to oriented graphs with bounded independence number, and to relational structures consisting of a tournament augmented by arbitrary binary relations, see the full version of this paper.

## 1.2 Overview of the proof

A fundamental idea regarding twin-width is that upper bounds on twin-width can be witnessed by orders on vertices which exclude grid-like structures in the adjacency matrix. This appears in the founding works of Guillemot and Marx [17] and Bonnet et al. [7], and the relation between twin-width and orders has been deeply explored in [6]. However it is difficult to witness *lower bounds* on twin-width with this approach: one needs to somehow prove that *all* orders contain grids. To this purpose, we want to construct in any tournament $T$ an order $<$ which, if $T$ has small twin-width, is a witness of this fact, i.e. $\mathrm{tww}(T, <) \leqslant f(\mathrm{tww}(T))$.

A tentative approach to obtain such an order is to describe it in FO logic. Indeed, FO transductions preserve twin-width up to some function [7, Theorem 39]. Thus, if $\Phi$ is a transduction which on any tournament $T$ gives some order $<$, then $\mathrm{tww}(T, <) \leqslant f(\mathrm{tww}(T))$ as desired. With a few additional requirements, such as $<$ being efficiently computable, it would be straightforward to obtain our results from the case of ordered graphs [6]. However this approach is impossible: to transduce a total order on the iterated lexicographic product of the 3-cycle with itself, one needs a first-order formula with size increasing in the number of iterations [3]. Remark that this counter-example has twin-width 1.

Instead, our approach is the following: we design a candidate total order $<$ on $T$, computable in polynomial time. If the bi-relation $(T, <)$ has small twin-width, we are done. On the other hand, if $(T, <)$ has large twin-width, then its adjacency matrix w.r.t. $<$ must contain a large high-rank grid by [6]. We then extract a subtournament $T' \subset T$ which still has a substantial (but logarithmically smaller) high-rank grid, and in which $<$ is roughly described by a FO transduction. This is enough to witness that $T$ has large twin-width. Using Ramsey arguments, we extract from $T'$ an obstruction $\mathcal{F}_=$, $\mathcal{F}_\leqslant$, or $\mathcal{F}_\geqslant$. The construction of the order is remarkably simple: we consider a binary search tree (BST), i.e. a tree in which the left, resp. right, branch of a node $x$ consists only of in-, resp. out-neighbours of $x$. The order $<$ is the left-to-right order on nodes of the tree. To summarize, the crucial property is

▶ **Lemma 1.4.** *There is a function $f$ such that for any tournament $T$ and BST order $<$ on $T$,* $\mathrm{tww}(T, <) \leqslant f(\mathrm{tww}(T))$*.*

Lemma 1.4 implies Theorem 1.3: to approximate the twin-width of $T$, it suffices to compute any BST order, which takes polynomial time, and then apply the approximation algorithm for ordered structures [6, Theorem 2], which is FPT. This last algorithm produces either a contraction sequence (which is valid for $(T, <)$ and a fortiori for $T$), or a witness that $(T, <)$ has large twin-width, which in turn implies that $T$ has large twin-width by Lemma 1.4.

Our main technical result is about extracting the obstructions $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$.

▶ **Theorem 1.5.** *Let $\mathcal{T}$ be a hereditary class of tournaments with the property that there are tournaments $T \in \mathcal{T}$ and BST orders $<$ such that* $\mathrm{tww}(T, <)$ *is arbitrarily large. Then $\mathcal{T}$ contains one of the classes $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$ as a subclass.*

Finally, the classes $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$ are complex in all the senses considered by Theorem 1.1.

▶ **Theorem 1.6.** *For each $R \in \{=, \leqslant, \geqslant\}$, the class $\mathcal{F}_R$*

1. *has unbounded twin-width;*
2. *contains at least $(\lfloor \frac{n}{2} \rfloor - 1)!$ tournaments on $n$ vertices counted up to isomorphism;*
3. *contains at least $(\lfloor \frac{n}{2} \rfloor - 1)! \cdot n!$ tournaments on vertex set $\{1, \dots, n\}$ counted up to equality;*
4. *efficiently interprets the class of all graphs;*
5. *and has AW[$*$]-hard FO model checking problem.*

Theorems 1.5 and 1.6 together imply Theorem 1.2. They also imply Lemma 1.4 when applied to the class of tournaments with twin-width at most $k$: this class cannot contain any of $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$, hence its tournaments must still have bounded twin-width when paired with BST orders. Finally, Theorems 1.2 and 1.6 directly imply that if $\mathcal{T}$ is a hereditary class with unbounded twin-width, then $\mathcal{T}$ satisfies none of the conditions of Theorem 1.1. The remaining implications of Theorem 1.1 – that is, when $\mathcal{T}$ has bounded twin-width, all other conditions hold – follow from known results on twin-width. By [7, Theorem 1], FO model checking has an FPT algorithm when a witness of bounded twin-width is given. Combined with Theorem 1.3, this gives an FPT algorithm for classes of tournaments with bounded twin-width. By [7, Theorem 39], a class of structures with bounded twin-width cannot transduce all graphs. Finally, by [8, Corollary 7.3], a class of structures with bounded twin-width contains at most $c^n$ structures on $n$ vertices up to isomorphism, for some constant $c$.

## 1.3 Context and related parameters

It is interesting to compare twin-width to other classical complexity measures for tournaments. Bounded twin-width implies bounded VC-dimension, since classes with unbounded VC-dimension contain all possible bipartite subgraphs, which is against single-exponential growth. *Cutwidth* was introduced by Chudnovsky, Fradkin and Seymour [11] to study tournament immersions. Bounded cutwidth is certified by a vertex ordering which can be shown to exclude grids, hence it is also a witness of bounded twin-width. Another parameter, closely related to subdivisions in tournaments, is *pathwidth*, studied by Fradkin and Seymour [14]. Bounded pathwidth of tournaments implies bounded cliquewidth, which in turn also implies bounded twin-width, see [7]. Thus, we have the following chain of inclusions (if a parameter is bounded, all the ones listed after are also bounded): cutwidth, pathwidth, cliquewidth, twin-width, and VC-dimension. For more on the subject, see Fomin and Pilipczuk [13, 21].

Regarding the binary search tree method for ordering tournaments, it corresponds to the KwikSort algorithm of Ailon, Charikar and Newman for approximating the minimum feedback arc set [1]. A difference is that their result requires the BST to be randomly chosen, whereas arbitrary BST provide approximations of twin-width.

## 1.4 Organisation of the paper

Section 2 summarises our definitions and notations. In section 3 the classes $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$ of obstructions to twin-width are defined, and we prove Theorem 1.6. Section 4 defines binary search trees, the associated orders, and some related notions. We then prove a crucial lemma which, from a partition into intervals of a BST order, extracts some FO definable substructure. Section 5 proves Lemma 1.4 using the former lemma, combined with results of [6]. See the extended version of this paper [16] for the full proof of Theorem 1.5, which builds on that of Lemma 1.4.

## 2 Preliminaries

This section summarizes the notions and notations used in this work. For $n \in \mathbb{N}$, we denote by $[n]$ the interval of integers $\{1, \dots, n\}$.

### 2.1 Tournaments, relational structures

A *tournament* $T$ consists of a set of vertices $V(T)$, and for each $u \neq v \in V(T)$, an arc oriented either $u \to v$ or $v \to u$ (but not both). If $x \in V(T)$, then $N^+(x) = \{y \mid x \to y\}$ and $N^-(x) = \{y \mid y \to x\}$ are the *in-* and *out-neighbourhood* respectively. A tournament is *transitive* if it contains no directed cycle, in which case it defines a total order on its vertices. We call *chain* a subset $X \subset V(D)$ which induces a transitive tournament.

Relational structures generalise graphs and tournaments. A relational *signature* is a finite set $\Sigma$ of *relation symbols* $R$, each with an arity $r \in \mathbb{N}$. A $\Sigma$-structure consists of a domain $A$ (vertices), and for each symbol $R \in \Sigma$ of arity $r$, an interpretation $R^S \subseteq A^r$ (hyperedges). E.g., tournaments and graphs are structures over a signature with a single binary relation. We restrict ourselves to *binary* structures, i.e. where all relation symbols have arity 2. An *ordered structure* $S$ is a structure over a relation $\Sigma$ with a special symbol $<$, whose interpretation $<^S$ is a total order on the domain of $S$.

If $S$ is a structure with domain $A$ and $B \subseteq A$, the substructure $S[B]$ *induced* by $B$ has domain $B$, and interprets each relation $R$ as the restriction of $R^S$ to $B$. All classes of structures considered here are *hereditary*, i.e. closed under induced substructures.

### 2.2 Matrices

A matrix is a map $M : R \times C \to \Gamma$, where $R, C$ are the *ordered sets* of rows and columns of the matrix, and $\Gamma$ and its alphabet (usually, $\Gamma = \{0, 1\}$). A submatrix of $M$ is the restriction of $M$ to some subsets of rows and columns. A *division* $\mathcal{D}$ of $M$ consist of partitions $\mathcal{R}, \mathcal{C}$ of the rows and columns respectively into *intervals*. It is a $k$-division if the partitions have $k$ parts each. A *cell* of the division is the submatrix induced by $X \times Y$ for some $X \in \mathcal{R}, Y \in \mathcal{C}$. A *$k$-grid* in a 0,1-matrix is a division in which every cell contains a "1".

For a tournament $T$ and a total order $<$ on $V(T)$, the adjacency matrix $A_{(T,<)}$ has $V(T)$ ordered by $<$ as rows and columns, and contains a "1" at position $(u, v)$ if and only if $u \to v$. This generalises to binary structures over any signature $\Sigma$, with $\{0, 1\}^\Sigma$ as alphabet.

### 2.3 Orders, Quasi-orders

A *quasi-order* $\preceq$ is a reflexive and transitive binary relation. The associated equivalence relation is $x \sim y$ iff $x \preceq y \wedge y \preceq x$. The strict component of the quasi-order is $x \prec y$ iff $x \preceq y$ and $y \not\preceq x$. The quasi-order is *total* if for all $x, y$, either $x \preceq y$ or $y \preceq x$. An *interval* of a quasi-order $\preceq$ is a set of the form $\{z \mid x \preceq z \preceq y\}$ for some $x, y$, called *endpoints*. An interval is a union of equivalence classes of $\sim$. Two subsets $X, Y$ are *overlapping* if there exist $x_1, x_2 \in X$ and $y_1, y_2 \in Y$ such that $x_1 \preceq y_1$ and $x_2 \succeq y_2$. Equivalently, $X, Y$ are non-overlapping iff there are disjoint intervals $I_X$,$I_Y$ such that $X \subseteq I_X$ and $Y \subseteq I_Y$.

### 2.4 Permutations

We denote by $\mathfrak{S}_n$ the group of permutations on $n$ elements. The *permutation matrix* $M_\sigma$ has a "1" at position $(i, j)$ if and only if $j = \sigma(i)$. A permutation $\tau$ is a *pattern* of $\sigma$ if $M_\tau$ is a submatrix of $M_\sigma$. We say that $\sigma$ contains a $k$-grid if $M_\sigma$ contains a $k$-grid. When this is the case, any permutation in $\mathfrak{S}_k$ is a pattern of $\sigma$. For example, the permutation $\sigma$ on $k^2$ elements defined by $\sigma(ki + j + 1) = kj + i + 1$ for any $0 \leqslant i, j < k$ contains a $k$-grid.

A permutation can be represented as a *bi-order*, i.e. the superposition of two total orders. Precisely, for $\sigma \in \mathfrak{S}_n$, the structure $\mathcal{O}_\sigma$ has domain $[n]$, and has for relations the natural order $<$, and the permuted order $<_\sigma$ defined as $i <_\sigma j$ if and only if $\sigma(i) < \sigma(j)$. Any bi-order is isomorphic to some $\mathcal{O}_\sigma$. Remark that $\tau$ is a pattern of $\sigma$ if and only if $\mathcal{O}_\tau$ is isomorphic to an induced substructure of $\mathcal{O}_\sigma$. We write $\mathcal{O}_{\mathfrak{S}}$ for the class of all finite bi-orders.

## 2.5 Twin-width

Twin-width, denoted e.g. $\mathrm{tww}(G)$, is a complexity parameter defined on graphs, and more generally on binary structures. We refer the reader to [7] for the definition, based on contraction sequences – it will not be used in this work. Instead, we rely on the following characterisation by grid-like structures in adjacency matrices. Recall that a division of a matrix is a partition of rows and columns into intervals. We say that a matrix is *$k$-diverse* if it contains at least $k$ different rows and $k$ different columns – which is equivalent to having rank at least $k'$ up to single-exponential bounds. Then, a *rank-$k$ division* is a $k$-division in which every cell is $k$-diverse. Bonnet et al. proved

▶ **Theorem 2.1** ([6, Theorem 2]). *There are functions $f, g$ such that for any graph (or binary structure) $G$ and any order $<$ on $V(G)$,*
- *if $\mathrm{tww}(G, <) \geqslant f(k)$ then the matrix $A_{(G,<)}$ has a rank-$k$ division, and*
- *if the matrix $A_{(G,<)}$ has a rank-$g(k)$ division, then $\mathrm{tww}(G, <) \geqslant k$.*

*Furthermore there is an FPT algorithm which given $G$, $<$, and $k$, finds either a rank-$k$ division in $A_{(G,<)}$ or a contraction sequence of width $f(k)$ for $(G, <)$.*

## 2.6 First-order logic

Recall from the introduction that we are interesting in FO Model Checking: given as input a structure $S$ and a first-order formula $\phi$, test if $S \models \phi$. We consider the complexity of this problem parametrized by the size $|\phi|$. In general, this problem is AW[$*$]-complete.

▶ **Theorem 2.2** ([12]). *FO Model Checking is AW[$*$]-complete on the class of all graphs.*

On the other hand, FO model checking is FPT for classes of structures with bounded twin-width, as long as a witness of twin-width is given.

▶ **Theorem 2.3** ([7, Theorem 1]). *Given a binary structure $S$ on $n$ vertices, a contraction sequence of width $k$ for $S$, and a FO formula $\phi$, one can test if $S \models \phi$ in time $f(k, \phi) \cdot n$.*

*Interpretations* are transformations of structures described using logical formulæ. For two relational signatures $\Sigma, \Delta$, a FO interpretation $\Phi$ from $\Sigma$ to $\Delta$ consists of, for each relation $R \in \Delta$ of arity $r$, a FO formula $\phi_R(x_1, \ldots, x_r)$ over the language $\Sigma$, and one last formula $\phi_{dom}(x)$ again over $\Sigma$. If $S$ is a $\Sigma$-structure, the result $\Phi(S)$ is obtained by
- choosing the same domain as $S$,
- interpreting $R \in \Delta$ as $\{(v_1, \ldots, v_r) \mid S \models \phi_R(v_1, \ldots, v_r)\}$, the tuples satisfying $\phi_R$,
- and finally taking the substructure induced by $\{v \mid S \models \phi_{dom}(v)\}$.

For instance, the square of a graph $G$ has the same vertices as $G$, with an edge $xy$ iff the distance of $x$ and $y$ in $G$ is at most 2. This is a FO interpretation with edges defined by

$$\phi(x, y) = E(x, y) \vee (\exists z.\ E(x, z) \wedge E(z, y))$$

where $E(x, x)$ denotes adjacency. The domain formula just "true" since we do not wish to delete vertices in this case. FO interpretations are closed under composition.

*Transductions* generalise interpretation with a non-deterministic coloring step. Let $\Sigma^+$ be the signature obtained by adding $r$ new *unary* relations $C_1, \dots, C_r$ to $\Sigma$. If $S$ is a $\Sigma$-structure, we denote by $S^+$ the set of $\Sigma^+$-structures obtained from $S$ by choosing an arbitrary interpretation of each $C_i$ as a subset of $V(S)$. Now a FO transduction $\Phi : \Sigma \to \Delta$ is described by the choice of $\Sigma^+$ augmenting $\Sigma$ with unary relations, and a FO interpretation $\Phi_I$ from $\Sigma^+$ from $\Delta$. The result of $\Phi$ is the set of $\Delta$-structures $\Phi(S) = \{\Phi_I(T) \mid T \in S^+\}$. That is, the interpretation of the unary relations $C_1, \dots, C_r$ on $S$ are chosen non-deterministically, and then $\Phi_I$ is applied.[1] Like interpretations, transductions can be composed.

Given classes $\mathcal{C}, \mathcal{D}$ of structures, we say that $\mathcal{C}$ *interprets* (resp. *transduces*) $\mathcal{D}$ if there is a FO interpretation (resp. transduction) $\Phi$ such that $\Phi(\mathcal{C}) \supseteq \mathcal{D}$. We furthermore say that $\mathcal{C}$ *efficiently interprets* $\mathcal{D}$ if there is also an algorithm which given as input $D \in \mathcal{D}$, finds in polynomial time some $C \in \mathcal{C}$ such that $\Phi(C) = D$. It is straightforward to show that this additional condition gives an FPT reduction for model checking.

▶ **Lemma 2.4.** *If $\mathcal{C}$ efficiently interprets $\mathcal{D}$, then there is an FPT reduction from FO Model Checking on $\mathcal{D}$ to FO Model Checking on $\mathcal{C}$.*

Recall that $\mathcal{O}_{\leqslant}$ denotes the class of bi-orders, which are encodings of permutations. The following is a folklore result, see e.g. [6, Lemma 10] for a very similar claim.

▶ **Lemma 2.5.** *The class $\mathcal{O}_{\leqslant}$ of bi-orders efficiently interprets the class of all graphs.*

Thus, using Lemma 2.4 and Theorem 2.2, FO Model Checking on $\mathcal{O}_{\leqslant}$ is AW[$*$]-complete.

FO transductions also preserve twin-width, up to some function.

▶ **Theorem 2.6** ([7, Theorem 39])**.** *If $\mathcal{S}$ is a class of binary structures with bounded twin-width and $\Phi$ is a FO transduction defined on $\mathcal{S}$, then $\Phi(\mathcal{S})$ also has bounded twin-width.*

A class of structures $\mathcal{S}$ is said to be *monadically NIP* if $\mathcal{S}$ does not transduce the class of all graphs. Theorem 2.6 implies that classes with bounded twin-width are monadically NIP. The weaker notion of (non-monadically) NIP also exists, however Braunfeld and Laskowski recently proved that NIP and monadically NIP are equivalent for hereditary classes [10].

## 2.7 Enumeration

A class $\mathcal{S}$ of graphs (or binary relational structures) is *small* if there exists $c$ such that $\mathcal{S}$ contains at most $c^n \cdot n!$ structures on the vertex set $[n]$. For instance, the class of trees is small, and more generally proper minor closed classes of graphs are small as shown by Norine et al. [20]. This was further generalised to classes of bounded twin-width by Bonnet et al.

▶ **Theorem 2.7** ([5, Theorem 2.4])**.** *Classes of structures with bounded twin-width are small.*

# 3 Forbidden classes of tournaments

This section defines the three minimal classes $\mathcal{F}_=$, $\mathcal{F}_\leqslant$, and $\mathcal{F}_\geqslant$ of obstructions to twin-width in tournaments. Each of them corresponds to some encoding of the class of all permutations. For $R \in \{=, \leqslant, \geqslant\}$ and any permutation $\sigma$, we will define a tournament $\mathcal{F}_R(\sigma)$. The class $\mathcal{F}_R$ is the hereditary closure of all $\mathcal{F}_R(\sigma)$.

[1] Additional operations such as duplication of vertices are often allowed in transductions, but these will not be needed in this work.

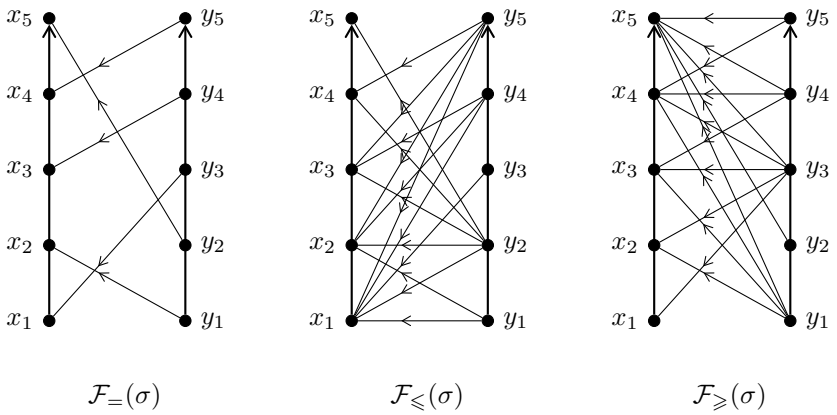



**Figure 1** The three classes of obstructions to twin-width in tournaments. For readability, edges oriented from some $x_i$ to some $y_j$ have been omitted. Each class consists of some encoding of the class of all permutations, represented here with the permutation $\sigma = 31452$.

Let $R \in \{=, \leqslant, \geqslant\}$, and let $\sigma \in \mathfrak{S}_n$ be a permutation on $n$ elements. The tournament $\mathcal{F}_R(\sigma)$ consists of $2n$ vertices, called $x_1, \dots, x_n, y_1, \dots, y_n$. Let $X = \{x_1, \dots, x_n\}$ and $Y = \{x_1, \dots, y_n\}$. Each of $X, Y$ is a chain under the natural order, i.e. there is an edge from $x_i$ to $x_j$, resp. from $y_i$ to $y_j$, if and only if $i < j$. The edges between $X$ and $Y$ encode $\sigma$ in a way specified by the relation $R$: there is an edge oriented from $y_j$ to $x_i$ if and only if $i \, R \, \sigma^{-1}(j)$. See Figure 1 for an example.

Thus in $\mathcal{F}_{=}(\sigma)$ the edges oriented from $Y$ to $X$ form a matching which encodes $\sigma$. In $\mathcal{F}_{\leqslant}(\sigma)$ and $\mathcal{F}_{\geqslant}(\sigma)$, these edges form a half-graph which orders $X$ and $Y$ by inclusion of neighbourhoods, so that the order on $X$ is the natural one, and the order on $Y$ encodes $\sigma$. Precisely, in $\mathcal{F}_{\geqslant}(\sigma)$, for any $i, j \in [n]$, we have

$$\big(N^-(x_i) \cap Y\big) \subseteq \big(N^-(x_j) \cap Y\big) \iff i \leqslant j \tag{1}$$

$$\text{and} \quad \big(N^-(y_i) \cap X\big) \subseteq \big(N^-(y_j) \cap X\big) \iff \sigma^{-1}(i) \leqslant \sigma^{-1}(j), \tag{2}$$

while in $\mathcal{F}_{\leqslant}(\sigma)$, the direction of inclusions is reversed.

▶ **Lemma 3.1.** *For each $R \in \{=, \leqslant, \geqslant\}$, the class $\mathcal{F}_R$ efficiently interprets the class $\mathcal{O}_{\mathfrak{S}}$ of bi-orders. Precisely, there is an interpretation $\Phi_R$, and for any permutation $\sigma \in \mathfrak{S}_n$, $n \geqslant 2$, there is a $\sigma' \in \mathfrak{S}_{n+1}$ computable in polynomial time such that $\mathcal{O}_\sigma = \Phi_R(\mathcal{F}_R(\sigma'))$.*

**Proof.** We will first show that $\mathcal{F}_R(\sigma)$ transduces $\mathcal{O}_\sigma$, and then how to remove the coloring step of the transduction by slightly extending $\sigma$.

Let $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_n\}$ be as in the definition of $\mathcal{F}_R(\sigma)$. The transduction uses coloring to guess the set $X$. It then defines two total orders on $Y$, which together describe $\sigma$. The first ordering is given by the direction of edges inside $Y$. The second depends on $R$:

- If $R$ is $=$, edges oriented from $Y$ to $X$ are a perfect matching. The direction of edges in $X$, interpreted through this matching, defines the second order on $Y$.
- If $R$ is $\geqslant$ or $\leqslant$, the second order is inclusion, respectively inverse inclusion, of in-neighbourhoods intersected with $X$, see (2).

With the knowledge of which subset is $X$, each of these orders is clearly definable with a first-order formula. Finally, the transduction deletes vertices of $X$, leaving only $Y$ and the two orders which encode $\sigma$.

Let us now show how to deterministically define the partition $X, Y$, at the cost of extending $\sigma$ with one fixed value. Here, we assume $n \geqslant 2$.

- If $R$ is $=$, define $\sigma' \in \mathfrak{S}_{n+1}$ by $\sigma'(n+1) = n+1$ and $\sigma'(i) = \sigma(i)$ for any $i \leqslant n$. Then, in $\mathcal{F}_=(\sigma')$, the unique vertex with out-degree 1 is $y_{n+1}$. Its out-neighbour is $x_{n+1}$, which verifies $X = N^-(x_{n+1}) \cup \{x_{n+1}\} \setminus \{y_{n+1}\}$.
- If $R$ is $\leqslant$, define $\sigma'(1) = n+1$ and $\sigma'(i+1) = \sigma(i)$. Then $y_{n+1}$ is the unique vertex with out-degree 1, and its out-neighbour is $x_1$, which satisfies $X = N^+(x_1) \cup \{x_1\}$.
- If $R$ is $\geqslant$, we once again define $\sigma'(1) = n+1$ and $\sigma'(i+1) = \sigma(i)$. Then $x_1$ has in-degree 1, and its in-neighbour is $y_{n+1}$. The only other vertex which may have in-degree 1 is $y_1$, and this happens if and only if $\sigma'(2) = 1$. When this is the case, the direction of the edge $x_1 \to y_1$ still allows to distinguish $x_1$ in FO logic. Then, having defined $x_1$, we obtain $y_{n+1}$ as its in-neighbour, which satisfies $X = N^+(y_{n+1})$.

In all three cases, $\mathcal{F}_R(\sigma')$ contains two extra vertices compared to $\mathcal{F}_R(\sigma)$. These extra vertices can be uniquely identified in first-order logic, and can then be used to define $X$. Combined with the previous transduction, this gives an interpretation of $\mathcal{O}_\sigma$ in $\mathcal{F}_R(\sigma')$. ◀

We can now prove that the classes $\mathcal{F}_R$ are complex.

▶ **Theorem 1.6.** *For each $R \in \{=, \leqslant, \geqslant\}$, the class $\mathcal{F}_R$*

1. *has unbounded twin-width;*
2. *contains at least $(\lfloor \frac{n}{2} \rfloor - 1)!$ tournaments on $n$ vertices counted up to isomorphism;*
3. *contains at least $(\lfloor \frac{n}{2} \rfloor - 1)! \cdot n!$ tournaments on vertex set $\{1, \dots, n\}$ counted up to equality;*
4. *efficiently interprets the class of all graphs;*
5. *and has AW[$*$]-hard FO model checking problem.*

**Proof.** Item 4 is straightforward by Lemmas 2.5 and 3.1, since efficient interpretations can be composed. By Lemma 2.4 and Theorem 2.2, this in turn implies Item 5. Item 3 implies Item 2 by a simple counting argument: in an isomorphism class, there are at most $n!$ choices of labelling of vertices with $\{1, \dots, n\}$ (less if there are automorphisms). Furthermore, each of Items 3 and 4 implies Item 1, by Theorem 2.7 and Theorem 2.6 respectively. Thus it only remains to prove Item 3.

By Lemma 3.1, for any permutation $\sigma \in \mathfrak{S}_n$ there is some $\mathcal{F}_R(\sigma')$ on $2n+2$ vertices such that $\Phi_R(\mathcal{F}_R(\sigma')) = \sigma$, where $\Phi_R$ is a fixed interpretation. Since interpretations preserve isomorphism, it follows that there are at least $n!$ non-isomorphic tournaments on $2n+2$ vertices in $\mathcal{F}_R$. Furthermore, the arguments of Lemma 3.1, it is easy to show that these $\mathcal{F}_R(\sigma')$ have no non-trivial automorphism. Thus, there are exactly $(2n+2)!$ distinct labellings of $\mathcal{F}_R(\sigma')$ with $\{1, \dots, 2n+2\}$. In total, this gives $(2n+2)! \cdot n!$ distinct graphs on vertices $\{1, \dots, 2n+2\}$ in $\mathcal{F}_R$, proving Item 3. ◀

Thus the classes $\mathcal{F}_=, \mathcal{F}_\leqslant, \mathcal{F}_\geqslant$ are obstructions to fixed parameter tractability of FO model checking and twin-width. The rest of the paper shows that they are the only obstructions. One may also verify that all three are minimal, i.e. none of them is contained in another.

## 4 Binary search tree orders

This section presents the good order for twin-width in tournaments. It is based on *binary search trees* (BST), which we define in a tournament $T$ as a rooted ordered binary tree $S$ (meaning that each node has a left and right child, either of which may be missing), whose nodes are the vertices of $T$, and such that for any $x \in S$

- the left child of $x$ (if any) and its descendants are in $N_T^-(x)$, and
- the right child of $x$ (if any) and its descendants are in $N_T^+(x)$, see Figure 2.

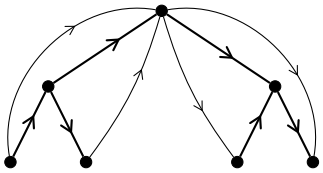


**Figure 2** A binary search tree in a tournament. The direction of omitted edges is not constrained.

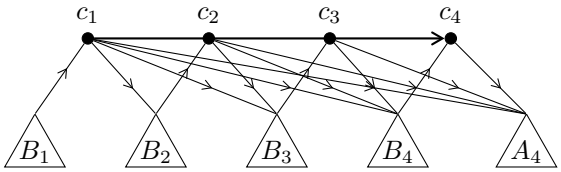



**Figure 3** Example of construction of the quasi-order $\preceq_C^+$. The quasi-order is from left to right, and the triangles are equivalence classes. The direction of omitted edges (from $B_i$ to $B_j \cup \{c_j\}$ for $i < j$) is not constrained. For $\preceq_C^-$, the direction of all edges would be reversed.

The *order* associated to $S$, denoted $<_S$, is the left-to-right order, i.e. the one which places a node $x$ after its left child and its descendants, but before its right child and its descendants. Such an order is called a *BST order*.

Remark that because $T$ is only a tournament and not an order as in a standard BST, there is no restriction on the direction of edges between the left and right subtrees of $x$. On the other hand, if $x$ is an ancestor of $y$, then there is an edge oriented from $x$ to $y$ if and only if $x <_S y$. Thus we have

▶ Remark 4.1. In a tournament $T$, any branch $B$ of a BST $S$ forms a chain which coincides with $<_S$. That is, for $x, y \in B$, the edge in $T$ is oriented from $x$ to $y$ if and only if $x <_S y$.

We will now define *chain quasi-orders*, which are FO definable quasi-orders with which we will approximate BST orders. Let $C$ be a chain in $T$. Its *chain quasi-order* $\preceq_C^+$ is defined as follows. Enumerate the vertices of $C$ as $c_1, \dots, c_k$ so that edges are oriented from $c_i$ to $c_j$ when $i < j$. Define $A_i = \bigcap_{j \leqslant i} N^+(c_j)$, and $B_i = A_{i-1} \cap N^-(c_i)$. Then each of $B_1, \dots, B_k$ and $A_k$ is an equivalence class of $\preceq_C^+$, and the classes are ordered as

$$B_1 \prec_C^+ c_1 \prec_C^+ B_2 \prec_C^+ c_2 \prec_C^+ \dots B_k \prec_C^+ c_k \prec_C^+ A_k,$$

see Figure 3. This can be seen as the left-to-right order of a partial BST consisting only of a single branch $c_1, \dots, c_k$, with $c_1$ as root and $c_k$ as leaf. It is also a coarsening of the lexicographic order: the latter would refine the order inside each class $B_i$ using $c_{i+1}, \dots, c_k$.

The dual quasi-order $\preceq_C^-$ is defined in the same way, but reversing the direction of all edges. Thus, we now enumerate $C$ so that edges are from $c_i$ to $c_j$ when $i > j$, while $A_i = \bigcap_{j \leqslant i} N^-(c_i)$ and $B_i = A_{i-1} \cap N^+(c_i)$. The rest of the definition is the same.

▶ **Lemma 4.2.** *There is a first-order transduction $\Phi$ which non-deterministically computes any chain quasi-order. That is, for any tournament $T$ and chain quasi-order $\preceq_C^o$, $(T, \preceq_C^o) \in \Phi(T)$.*

**Proof.** The transduction first guesses $C$ and $o$, and obtains the order within $C$ from the edges of $T$. It is then simple to express the definition of $\preceq_C^o$ in first-order logic. ◀

We now prove our main technical lemma on BSTs, which shows that BST orders can to some extent be approximated by chain quasi-orders.

▶ **Lemma 4.3.** *Let $T$ be a tournament and $S$ be a BST with order $<_S$. There is a function $f(k) = 2^{O(k)}$ independent of $T$ and $S$ such that for any family $\mathcal{P}$ of at least $f(k)$ disjoint intervals of $<_S$, there is a chain $C$ in $T$, an orientation $o \in \{+,-\}$ and a subfamily $\mathcal{P}' \subset \mathcal{P}$ such that $|\mathcal{P}'| \geqslant k$ and such that the intervals of $\mathcal{P}'$ are non-overlapping for $\preceq_C^o$.*

*Furthermore, $C$, $o$, and $\mathcal{P}'$ can be computed in linear time.*

**Proof.** Let $T$ be a tournament, $S$ a BST of $T$ and $<_S$ the corresponding order. Let $\mathcal{P}$ be a family of at least $f(k)$ disjoint intervals of $<_S$, where $f(k) = 2^{O(k)}$ will be determined later.

We choose a branch $B = b_0, \dots, b_p$ of $S$ by the following process. First $b_0$ is the root of $S$. For each (yet to be determined) $b_i$, let $S_i$ be the subtree of $S$ rooted in $b_i$, and define the weight $w_i$ to be the number of classes of $\mathcal{P}$ intersected by $S_i$. Then $b_{i+1}$ is chosen to be the child of $b_i$ which maximizes $w_{i+1}$. This choice ensures that

$$2w_{i+1} + 1 \geqslant w_i. \tag{3}$$

For each $i < p$, let $d_i$ be the child of $b_i$ other than $b_{i+1}$ (sometimes $d_i$ does not exist), and let $D_i$ be the subtree of $S$ rooted at $d_i$ ($D_i$ is empty if $d_i$ does not exist). Furthermore, let $L, R$ be the sets of vertices which are before, resp. after the leaf $b_p$ in the order $<_S$. For any $0 \leqslant i \leqslant j \leqslant p$, let

$$L_{i,j} \stackrel{\text{def}}{=} \bigcup_{\substack{i \leqslant \ell < j \\ b_\ell \in L}} \{b_\ell\} \cup D_\ell, \qquad \text{and} \qquad R_{i,j} \stackrel{\text{def}}{=} \bigcup_{\substack{i \leqslant \ell < j \\ b_\ell \in R}} \{b_\ell\} \cup D_\ell.$$

Roughly speaking, $L_{i,j}$, resp. $R_{i,j}$ consists of subtrees branching out of $B$ on the left, resp. right, between $b_i$ and $b_j$.

▷ Claim 4.4. For any $i, j$, the subtree $S_i$ is partitioned into $L_{i,j} <_S S_j <_S R_{i,j}$.

Proof. Clearly $L_{i,j}, S_j, R_{i,j}$ partition $S_i$. Furthermore, if $\ell < j$ and $b_\ell \in L$, then $b_\ell <_S S_j$, and in turn $D_\ell <_S b_\ell$. This proves $L_{i,j} <_S S_j$, and symmetrically $S_j <_S R_{i,j}$. ◁

▷ Claim 4.5. For $0 \leqslant i < j \leqslant p$, if $w_i \geqslant w_j + 3$, then there is a part $P \in \mathcal{P}$ such that $P \subset L_{i,j}$ or $P \subset R_{i,j}$.

Proof. At least three parts of $\mathcal{P}$ intersect $S_i$ but not $S_j$. Since these three parts and $S_i$ are all intervals of $<_S$, one of these parts, say $P$, is contained in $S_i$. Thus $P$ is a subset of $S_i$ but does not intersect $S_j$, which by Claim 4.4 implies $P \subset L_{i,j}$ or $P \subset R_{i,j}$. ◁

Construct a sequence $i_0 < \cdots < i_{2k}$ of indices in $\{0, \dots, p\}$ inductively by taking $i_0 = 0$, and choosing $i_{\ell+1}$ minimal such that $w_{i_{\ell+1}} \leqslant w_{i_\ell} - 3$. Using (3) and the minimality of $i_{\ell+1}$, we obtain for all $\ell$ that $2w_{i_{\ell+1}} + 1 \geqslant w_{i_{\ell+1}-1} > w_{i_l} - 3$, hence

$$2w_{i_{\ell+1}} + 3 \geqslant w_{i_l}. \tag{4}$$

We can now define $f$ by $f(0) = 1$ and $f(k+1) = 4f(k) + 9$. By assumption, $w_0 = |\mathcal{P}| \geqslant f(k)$, and it follows from (4) that the construction of $i_\ell$ can be carried out up to $i_{2k}$.

Define $L'_\ell = L_{i_{\ell-1}, i_\ell}$, and similarly $R'_\ell = R_{i_{\ell-1}, i_\ell}$, see Figure 4. By Claim 4.5, for any $\ell \in [2k]$, either $L'_\ell$ or $R'_\ell$ contains a part of $\mathcal{P}$. Thus, either there are at least $k$ distinct $L'_\ell$ containing a part of $\mathcal{P}$, or there are at least $k$ distinct $R'_\ell$ containing a part of $\mathcal{P}$. Assume the former case without loss of generality. We will now forget the vertices which are not in $L$.

Define $C = L \cap B$. By Remark 4.1, this is a chain, whose order coincides with $<_S$. Furthermore, at any node $x$ of $C$, the branch $B$ does descend on the right side, since $x <_S b_p$. Thus, the order in $C$ also coincides with the ancestor-descendent order of $S$. (Remark here

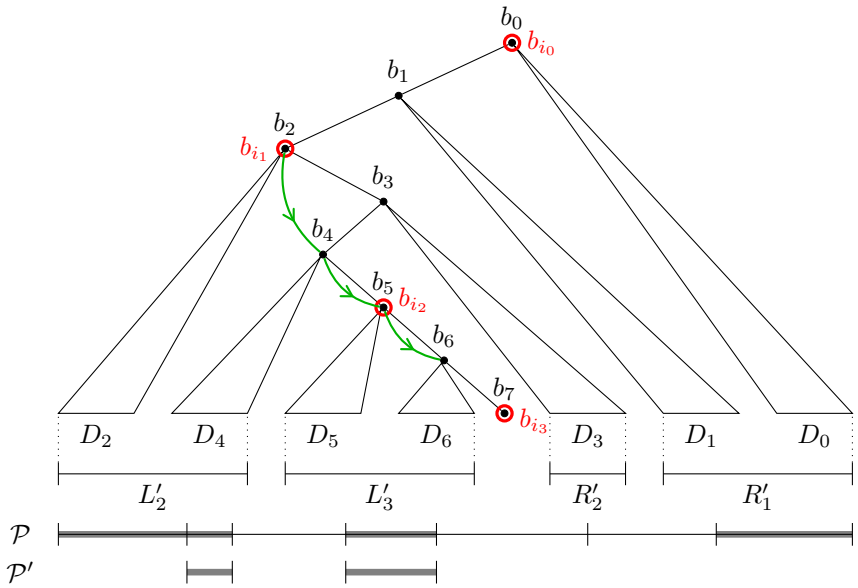



**Figure 4** Sketch of the proof of Lemma 4.3. In the upper half, the BST $T$ with the extracted branch $B$; circled in red, the extracted subsequence $b_{i_\ell}$; in green arrows, the chain $C = B \cap L = \{b_2, b_4, b_5, b_6\}$. Below the tree, from top to bottom: the partition in $L'_\ell$ and $R'_\ell$; the initial family (here partition) $\mathcal{P}$, with the parts contained in some $L'_\ell$ or $R'_\ell$ highlighted; the final family $\mathcal{P}'$, obtained by selecting a part of $\mathcal{P}$ inside each possible $L'_\ell$.

that if we were in $R$ instead of $L$, the order of $C$ would be the inverse of the ancestor-descendant order.) Now, if $C$ is enumerated as $c_0 <_S \cdots <_S c_t$, and $C_i$ is the subtree branching out on the left of $c_i$, defined similarly to $D_i$, then the chain quasi-order $\preceq^+_C$ restricted to $L$ is exactly

$$C_0 \prec^+_C c_0 \prec^+_C C_1 \prec^+_C c_1 \prec^+_C \dots \prec^+_C c_t$$

where each subtree $C_i$ is an equivalence class. (In $R$, we would instead use $\preceq^-_C$.) From this description, we obtain that any $L_{i,j}$ is an interval of $\preceq^+_C$ restricted to $L$.

For each $L'_\ell$, select a part of $\mathcal{P}$ included in $L'_\ell$ if any, and define $\mathcal{P}'$ as the collection of selected parts. Thus $\mathcal{P}' \subset \mathcal{P}$, and we know from the choice of the family $\{L'_\ell\}_{\ell \in [2k]}$ that $|\mathcal{P}'| \geqslant k$. Furthermore, if $X \neq Y$ are parts of $\mathcal{P}'$, there are $s \neq t$ such that $X \subseteq L'_s$ and $Y \subseteq L'_t$. Since each $L'_\ell$ is an interval of $(L, \preceq^+_C)$, this implies that $X$ and $Y$ are non-overlapping for $\preceq^+_C$. Thus $\mathcal{P}'$ satisfies all desired properties.

Finally, given the BST $S$ and the family $\mathcal{P}$, it is routine to compute the weights $w_i$ of all nodes in $S$ by a bottom-up procedure; this only requires to compute the left-most and right-most parts of $\mathcal{P}$ intersecting each subtree. From this, one can in linear time choose the branch $B$, the indices $i_\ell$, the better side $L$ or $R$, and finally compute $C$ and $\mathcal{P}'$. ◀

## 5 BST orders witness twin-width

In this section, we prove Lemma 1.4, i.e. that BST orders are good for twin-width. The proof heavily uses model-theoretic results from [6]. Due to space constraints, the combinatorial proof of the stronger result Theorem 1.5 is omitted, see the extended version of this paper [16].

If $\mathcal{T}$ is a class of tournaments, we denote by $\mathcal{T}^{BST}$ the class of ordered tournaments $(T, <_S)$ where $T \in \mathcal{T}$ and $<_S$ is the order of some BST $S$ on $T$. With this notation, Lemma 1.4 can be restated as

▶ **Lemma 5.1.** *If $\mathcal{T}$ is a hereditary class of tournaments with bounded twin-width, then $\mathcal{T}^{BST}$ also has bounded twin-width.*

**Proof.** Fix $\mathcal{T}$ a class of tournaments with twin-width at most $t$, and assume by contradiction that $\mathcal{T}^{BST}$ has unbounded twin-width. Then by Theorem 2.1, for any $k$ there is some $(T, <_S) \in \mathcal{T}^{BST}$ whose adjacency matrix contains a rank-$k$ division. That is, there are partitions $A_1, \dots, A_k$ and $B_1, \dots, B_k$ of $V(T)$ into intervals of $<_S$ such that the adjacency matrix of any $A_i$ versus $B_j$ is $k$-diverse.

If $k$ is chosen to be $k = f(\ell)$ where $f$ is the function of Lemma 4.3, then we obtain two chain quasi-orders $\preceq^A, \preceq^B$ in $T$, and subfamilies $A_{i_1}, \dots, A_{i_\ell}$ and $B_{j_1}, \dots, B_{j_\ell}$ which are non-overlapping for $\preceq^A$ and $\preceq^B$ respectively. We can in fact assume that $A_{i_1}, \dots, A_{i_\ell}$ are disjoint intervals of $\preceq_A$, by replacing them by their closure

$$\bar{A}_{i_t} \stackrel{\text{def}}{=} \{x \mid \exists y, z \in A_{i_t},\ y \preceq_A x \preceq_A z\}$$

Let $T^+$ be the structure $T$ augmented by the quasi-orders $\preceq_A$ and $\preceq_B$. In $T^+$, each interval $\bar{A}_{i_t}$ can be described by its two endpoints. Using the terminology of [6, section 9], this means that $\bar{A}_{i_1}, \dots, \bar{A}_{i_\ell}$ is a *definable disjoint family*. Naturally, the same holds for $\bar{B}_{j_1}, \dots, \bar{B}_{j_\ell}$. Finally, $A_i$ versus $B_j$ being $k$-diverse is a very special case of the model-theoretic notion of $A$ having $k$ *distinct* $\Delta$*-types over* $B$, when $\Delta$ consists only of the formula "being adjacent".

Let $\mathcal{T}^+$ denote the class of tournaments in $\mathcal{T}$ augmented by any two chain quasi-orders. We have just proved that for arbitrary large $k, \ell$, there are structures $T^+ \in \mathcal{T}^+$ containing two families of $\ell$ disjoint subsets $\left(\bar{A}_{i_t}\right)_{t\in[\ell]}$ and $\left(\bar{B}_{j_t}\right)_{t\in[\ell]}$ definable by a fixed formula, and such that each $\bar{A}_{i_t}$ has $k$ distinct $\Delta$-types over each $\bar{B}_{j_t}$. That is, the class $\mathcal{T}^+$ is *unrestrained* in the sense of [6, Definition 50]. By [6, Theorem 54], it follows that $\mathcal{T}^+$ is not monadically NIP, hence has unbounded twin-width. But it follows from Lemma 4.2 that $\mathcal{T}^+$ is obtained from $\mathcal{T}$ by a first-order transduction, contradicting that $\mathcal{T}$ has bounded twin-width. ◀

## References


1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), November 2008. doi:10.1145/1411509.1411513.

2 Noga Alon, Graham Brightwell, H.A. Kierstead, A.V. Kostochka, and Peter Winkler. Dominating sets in k-majority tournaments. *Journal of Combinatorial Theory, Series B*, 96(3):374–387, 2006. doi:10.1016/j.jctb.2005.09.003.

3 Mikołaj Bojańczyk. personal communication, July 2022.

4 Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: delineation and win-wins. *CoRR*, abs/2204.00722, 2022. doi:10.48550/arXiv.2204.00722.

5 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

6 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *CoRR*, abs/2102.03117, 2021, to appear at STOC 2022. arXiv:2102.03117.

7 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

8 Édouard Bonnet, Jaroslav Nesetril, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.

9 Youssef Boudabbous and Maurice Pouzet. The morphology of infinite tournaments; application to the growth of their profile. *European Journal of Combinatorics*, 31(2):461–481, 2010. Combinatorics and Geometry. doi:10.1016/j.ejc.2009.03.027.

10 Samuel Braunfeld and Michael C Laskowski. Existential characterizations of monadic nip. *arXiv preprint*, 2022. doi:10.48550/arXiv.2209.05120.

11 Maria Chudnovsky, Alexandra Fradkin, and Paul Seymour. Tournament immersion and cutwidth. *J. Comb. Theory, Ser. B*, 102:93–101, January 2012. doi:10.1016/j.jctb.2011.05.001.

12 Rodney G Downey, Michael R Fellows, and Udayan Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. *DMTCS*, 96:194–213, 1996.

13 Fedor V. Fomin and Michał Pilipczuk. On width measures and topological problems on semi-complete digraphs. *Journal of Combinatorial Theory, Series B*, 138:78–165, 2019. doi:10.1016/j.jctb.2019.01.006.

14 Alexandra Fradkin and Paul Seymour. Tournament pathwidth and topological containment. *Journal of Combinatorial Theory, Series B*, 103(3):374–384, 2013. doi:10.1016/j.jctb.2013.03.001.

15 Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Daniel Lokshtanov, and M. S. Ramanujan. A new perspective on fo model checking of dense graph classes. *ACM Trans. Comput. Logic*, 21(4), July 2020. doi:10.1145/3383206.

16 Colin Geniet and Stéphan Thomassé. First order logic and twin-width in tournaments and dense oriented graphs, 2022. arXiv:2207.07683.

17 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

18 Mithilesh Kumar and Daniel Lokshtanov. Faster Exact and Parameterized Algorithm for Feedback Vertex Set in Tournaments. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, volume 47 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2016.49.

19 Adam Marcus and Gábor Tardos. Excluded permutation matrices and the Stanley-Wilf conjecture. *J. Comb. Theory, Ser. A*, 107(1):153–160, 2004. doi:10.1016/j.jcta.2004.04.002.

20 Serguei Norine, Paul Seymour, Robin Thomas, and Paul Wollan. Proper minor-closed families are small. *Journal of Combinatorial Theory, Series B*, 96(5):754–757, 2006. doi:10.1016/j.jctb.2006.01.006.

21 Michał Pilipczuk. *Tournaments and Optimality: New Results in Parameterized Complexity.* PhD thesis, University of Bergen, August 2013. URL: https://bora.uib.no/bora-xmlui/bitstream/handle/1956/7650/dr-thesis-2013-Michal-Pilipczuk.pdf?sequence=1.

# Improved Approximation Algorithms for the Expanding Search Problem

**Svenja M. Griesbach**
Institute for Mathematics, Technische Universität Berlin, Germany

**Felix Hommelsheim**
Faculty of Mathematics and Computer Science, Universität Bremen, Germany

**Max Klimm**
Institute for Mathematics, Technische Universität Berlin, Germany

**Kevin Schewior**
Department Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

## Abstract

A searcher faces a graph with edge lengths and vertex weights, initially having explored only a given starting vertex. In each step, the searcher adds an edge to the solution that connects an unexplored vertex to an explored vertex. This requires an amount of time equal to the edge length. The goal is to minimize the weighted sum of the exploration times over all vertices. We show that this problem is hard to approximate and provide algorithms with improved approximation guarantees. For the general case, we give a $(2e + \varepsilon)$-approximation for any $\varepsilon > 0$. For the case that all vertices have unit weight, we provide a 2e-approximation. Finally, we provide a PTAS for the case of a Euclidean graph. Previously, for all cases only an 8-approximation was known.

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

**Keywords and phrases** Approximation Algorithm, Expanding Search, Search Problem, Graph Exploration, Traveling Repairperson Problem

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.54

**Related Version** *Full Version*: `https://arxiv.org/abs/2301.03638` [28]

**Funding** *Svenja M. Griesbach*: Supported by Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy, Berlin Mathematics Research Center (grant EXC-2046/1, Project 39068689) and HYPATIA.SCIENCE (Department of Mathematics and Computer Science, University of Cologne).
*Max Klimm*: Supported by Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy, Berlin Mathematics Research Center (grant EXC-2046/1, Project 390685689).
*Kevin Schewior*: Supported in part by the Independent Research Fund Denmark, Natural Sciences (grant DFF-0135-00018B).

**Acknowledgements** We thank Spyros Angelopoulos for fruitful discussions and pointers to earlier literature.

## 1 Introduction

A vital issue faced by disaster-relief teams sent to regions devastated by natural or man-made catastrophes is to decide where to search for buried or isolated people. The fundamental issues behind these decisions are that, in emergency situations, technical means for probing and for clearing areas are often limited, there is no full knowledge concerning the whereabouts of potential survivors, and rescue operations are time-critical since the chances of survival decrease with the time needed for rescue; see also the discussion in Averbakh and Pereira [13]. Mathematically, we model this problem using an undirected graph with edge lengths. The




vertices of the graph correspond to different locations in the disaster area, and the edges between them correspond to possible connections between the locations. The length of an edge corresponds to the time that is needed to clear the connection. Clearing a connection may mean to clear a road connection of rubble or explosives, or to dig in snow, dirt, or debris. There is a single rescue team initially located at a designated root vertex. Based on experience, the rescue team has knowledge about the number of survivors that is located at the different locations. The goal is to minimize the average time at which the survivors are reached.

A solution to the problem is given by a sequence of edges to clear until all vertices (with non-zero weight) can be reached. Once an edge is cleared, it can be traveled along in negligible time by the rescue team, so that only the time needed to clear edges is considered. A search problem of this kind is called *expanding search problem* (ESP) since the set of vertices accessible by the rescue team expands in every step. This is in contrast to *pathwise search problems* where the actual movement of the searcher is modeled and traversing an edge always requires time equal to the length of the edge, no matter whether it is the first traversal or not.

Generally speaking, expanding search problems are a suitable model when the time needed to traverse an edge for the first time is significantly higher than the time needed to traverse this edge any time after the first time, and, thus, the time needed for further movements can be neglected. Further applications of expanding search problems are in mining where the time needed to dig a new tunnel is much higher than moving via already dug tunnels to previously explored locations (Alpern and Lidbetter [4]) and when securing an area from a hidden explosive where the time needed to move within a safe region can be neglected compared to the time needed to secure a new area (Angelopoulos et al. [6]).

**Our contribution.** In this work, we provide polynomial-time approximation algorithms with improved approximation guarantees for ESP. We first give an approximation algorithm for the general case with arbitrary vertex weights with an approximation guarantee of $2\mathrm{e} + \varepsilon$ for any $\varepsilon > 0$ where $2\mathrm{e} \approx 5.44$ (Theorem 1). For the unweighted case where all vertices have the same weight, we provide an approximation algorithm with an approximation guarantee of $2\mathrm{e}$ (Theorem 6). The result for the unweighted case is obtained by concatenating $k$-minimum spanning trees ($k$-MSTs) for varying values of $k$ and of exponentially increasing length. Using the probabilistic method on lengths with random factor finally yields an additional factor of e. This technique has been used for pathwise search problems [16, 26]; we here adapt it to the case of expanding search. For the weighted case, instead of $k$-MSTs, we use the quota version of the $k$-MST problem where vertices have non-negative weights and, given $q \in \mathbb{N}$, the task is to find a length-minimal tree with vertex weight at least $q$. Johnson et al. [31] noted that any approximation algorithm for $k$-MST that relies on the Goemans–Williamson algorithm [27] for the prize-collecting Steiner-tree problem can be turned into an approximation algorithm for the quota version with the same approximation guarantee. We follow this line of reasoning and show that also the approximation algorithm of Garg [25] that relies on a modified version of the Goemans–Williamson algorithm can be turned into a 2-approximation algorithm for the quota version of the problem. Relying on this result, we solve the quota version for a polynomial number of quotas (thereby losing the factor of $1 + \varepsilon$) and use these solutions to construct a sequence of spanning trees of exponentially increasing length. Concatenating these solutions yields the claimed factor.

We then give a polynomial-time approximation scheme (PTAS) for the case of a Euclidean graph (Theorem 7). For this result, we use a decomposition approach by Sitters [40] for the pathwise search problem that relies on partitioning an instance into subinstances. A

central challenge when adapting this approach to the expanding search setting is that, unlike pathwise search, expanding search is not memoryless as points contained in one subinstance may be used as Steiner points in another subinstance. We address this difficulty by keeping such points in the subinstance with zero weight so that the partitions become overlapping. To obtain a PTAS for the subproblem, we adapt techniques developed by Arora [9] for Euclidean TSP. As a byproduct, we obtain a $(1+\varepsilon)$-approximate reduction from general ESP to unweighted ESP, implying an alternative $(2\mathrm{e}+\varepsilon)$-aproximation which may be of independent interest.

For all variants considered in this paper, i.e., the unweighted case, the weighted case, and the Euclidean case, the best approximation algorithm was an 8-approximation due to Hermans et al. [29].

Finally, we show that there is no PTAS for ESP unless $\mathsf{P}=\mathsf{NP}$ (Theorem 14). The proof follows a similar idea as the hardness proof for the traveling repairperson problem suggested in [15]. However, in comparison to the pathwise search, in our setting the solution needs to be structured. Showing this property turns out to be rather elaborate. Previously, it was only known that the expanding search problem is NP-hard [13]. Due to space constraints, we defer some proofs and more detailed descriptions to the full version of this paper [28].

**Further related work.** The unweighted pathwise search problem where all vertices have unit weight is also known as the *traveling repairperson problem*. Sahni and Gonzales [38] showed that the problem cannot be efficiently approximated within a constant factor unless $\mathsf{P}=\mathsf{NP}$ on complete non-metric graphs when the searcher is required to take a Hamiltonian tour. Afrati et al. [1] considered the problem in metric spaces and gave an exact algorithm with quadratic runtime when the metric is induced by a path. This can be improved to linear runtime as shown by García et al. [24]. Minieka [36] proposed an exact polynomial algorithm for the case that the metric is induced by an unweighted tree. Sitters [39] showed that the problem is NP-hard when the metric is induced by a tree with edge lengths 0 and 1.

The first approximation algorithm of the metric traveling repairperson problem is due to Blum et al. [15] who gave a 144-approximation. After a series of improvements [7, 8, 10, 26, 33], the best factor so far is a 3.59-approximation for general metrics [16], and a polynomial-time approximation scheme for trees [40] and on the Euclidean plane [40]. Further variants of the problem have been studied both in terms of exact solution methods and in terms of competitive algorithms, among other settings with directed edges [20, 21, 37], with processing times and time windows [43], with profits at vertices [18], with multiple searchers [17, 19, 35], and online variants [34]. The vertex-weighted version of the problem is often referred to as *the* pathwise search problem. It has been shown to be NP-hard in metric graphs by Trummel and Weisinger [42] and was further studied in [33]. The approximation schemes in [40] apply to the weighted case as well.

The expanding search problem has received considerably less attention in the literature than the pathwise problem. It has been shown to be NP-hard by Averbakh and Pereira [13]. Alpern and Lidbetter [4] introduced a polynomial-time algorithm for the case when the graph is a tree and gave heuristics for general graphs. Averbakh et al. [12] considered a generalization of the problem with multiple searchers when the underlying graph is a path; Tan et al. [41] considered multiple searchers in a tree network. The first constant-factor approximation for general metrics is the 8-approximation due to Hermans et al. [29], based on an exact algorithm on trees [4]. Angelopoulos et al. [6] studied the expanding search ratio of a graph. This value is defined as the minimum over all expanding searches of the maximum ratio of the time to reach a vertex by the search algorithm and the time to reach the same vertex by a shortest path. Angelopoulos et al. showed that this ratio is NP-hard to compute and gave a search strategy that achieves a $(4\ln 4)$-approximation of the optimum.

The pathwise and expanding search problems appear naturally as strategies of the seeker in a two-player zero sum game between hider and seeker where the hider chooses a vertex that maximizes the expected search time whereas the seeker aims to minimize the search time. Gal [22] computed the value (i.e., the unique search time in an equilibrium of the game) of the pathwise search game on a tree; Alpern and Lidbetter [4] computed this value for the expanding search game; see also [5] for approximations of this value for general graphs. For more details on search games, we refer to [2, 3, 23, 30, 32].

## 2 Preliminaries

We consider a connected undirected graph $G = (V, E)$ with $|V| = n$ and a designated root vertex $r \in V$. Every vertex $v \in V$ has a weight $w_v \in \mathbb{N} = \{0, 1, 2, \dots\}$, and we denote by $V^* \subseteq V$ the set of vertices with $w_v > 0$. Every edge $e \in E$ has a length $\ell_e \in \mathbb{N}$. We use $\mathbb{N}_{>0}$ when we refer to $\mathbb{N} \setminus \{0\}$. We consider an agent that is initially located at the root and performs an *expanding search pattern* $\sigma$. Such a pattern is given by a sequence of distinct edges $\sigma = (e_1, \dots, e_m)$ for some $m \leq n-1$ such that $r \in e_1$ and the set $\{e_1, \dots, e_i\}$ forms a tree in $G$ for all $i \in \{1, \dots, m\}$. For a vertex $v \in V^* \setminus \{r\}$, let $k_v(\sigma) = \inf\{i \in \{1, \dots, m\} : v \in e_i\}$ be the index of the first edge that contains $v$ and set $k_r(\sigma) = 0$. We then call $L_v(\sigma) = \sum_{i=1}^{k_v(\sigma)} \ell_{e_i}$ the *latency* of the vertex $v \in V^*$ under expanding search pattern $\sigma$. Our goal is to find an expanding search pattern $\sigma$ that minimizes the *total latency* $L(\sigma) = \sum_{v \in V^*} w_v L_v(\sigma)$. Note that vertices $v$ with $w_v = 0$ do not appear in the objective function, and, hence, do not need to be visited. They may, however, be used as Steiner vertices in the constructed search trees and, hence, cannot be contracted as in the pathwise search problem. When the pattern $\sigma$ is clear from context, we drop the dependency on $\sigma$ and simply write $L$, $L_v$, and $k_v$. The *length* $\ell(\sigma)$ of a search pattern $\sigma$ is given by the sum of edge costs, i.e., $\ell(\sigma) = \sum_{e \in \sigma} \ell_e$. Finally, for two expanding search patterns $\sigma = (e_1, \dots, e_m), \sigma' = (e'_1, \dots, e'_{m'})$, we denote by $\sigma + \sigma'$ their concatenation, i.e., the subsequence of $(e_1, \dots, e_m, e'_1, \dots, e'_{m'})$ in which any edge closing a cycle is skipped.

## 3 The weighted case

In this section, we consider the general case of the expanding search problem where the weights $w_v \in \mathbb{N}$ are arbitrary for all $v \in V \setminus \{r\}$, and $w_r = 0$. We prove the following theorem.

▶ **Theorem 1.** *For every $\varepsilon > 0$, there is a polynomial-time $(2\mathrm{e} + \varepsilon)$-approximation algorithm for the expanding search problem.*

The approximation algorithm that we devise in this section is based on the approximate solution of several quota versions of the prize-collecting Steiner tree problem. In this problem, we are given a connected undirected graph $G = (V, E)$ with designated root vertex $r \in V$, non-negative edge lengths $\ell_e \in \mathbb{R}_{\geq 0}, e \in E$, vertex weights $w_v \in \mathbb{N}$, $v \in V \setminus \{r\}$, and a quota $q \in [0, W]$, where $W \coloneqq \sum_{v \in V} w_v$. The task is to find a subgraph that is a tree $T = (V_T, E_T)$ such that $r \in V_T$ and $\sum_{v \in V_T} w_v \geq q$ minimizing $\ell(T) \coloneqq \sum_{e \in E_T} \ell_e$. We argue that this problem admits a 2-approximation. The proof can be found in the full version [28].

▶ **Lemma 2.** *For the quota version of the prize-collecting Steiner tree problem, a 2-approximation can be computed in polynomial time.*

To approximate ESP, fix $\varepsilon > 0$. For notational convenience, we show an approximation algorithm with guarantee $2(1+\varepsilon)\mathrm{e}$. We solve the quota problem for quotas $W - W(1+\varepsilon)^{-i}$ for all $i \in \{0, \dots, \omega\}$, where we let $\omega := \left\lceil \frac{\log W}{\log(1+\varepsilon)} \right\rceil$. Note that, for fixed $\varepsilon$, the number

$\omega$ is polynomial in the encoding length of the input. In this way, we obtain $\omega + 1$ trees $T_0, T_1, \dots, T_\omega$. By construction, tree $T_0$ has to collect a total weight of 0, so $T_0$ is the tree $T_0 = (\{r\}, \emptyset)$ consisting only of the root vertex. By the choice of $\omega$, the tree $T_\omega$ has to collect a total weight of $W - W(1+\varepsilon)^{-\omega} > W - 1$. This implies that the tree $T_\omega$ collects all weight since the weights are integers. We then construct a directed auxiliary graph $H = (V_H, A_H)$ with vertex set $V_H = \{0, \dots, \omega\}$ and arc set $A_H = \{(i,j) : i < j\}$. We set the cost of arc $(i,j)$ equal to $c_{i,j} = W(1+\varepsilon)^{-i}\ell(T_j)$. Next, we compute a shortest $(0,\omega)$-path $P = (n_0, \dots, n_l)$ with $n_0 = 0$ and $n_l = \omega$ for some $l \in \mathbb{N}$. We construct from this path an expanding search pattern with $l$ phases. In phase $j \in \{1, \dots, l\}$, we explore all edges in $e \in E[T_{n_j}]$ with $|e \cap (\bigcup_{i=0}^{j-1} V[T_{n_i}])| < 2$ in an order such that the subgraph of explored vertices is connected at all times. In that fashion, when phase $j$ is finished, all vertices in $V[T_{n_j}]$ have been explored. Since $n_l = \omega$ and $T_\omega$ collects the total weight $W$, all vertices $v$ with $w_v > 0$ have been explored when the algorithm terminates. Formally, the algorithm is given as follows:

1) For all $i \in \{0, 1, \dots, \omega\}$ solve the quota version of the prize-collecting Steiner tree problem with quota $q = W - W(1+\varepsilon)^{-i}$ with the 2-approximation algorithm of Lemma 2 and obtain $\omega + 1$ trees $T_0, T_1, \dots, T_\omega$.
2) Construct an auxiliary weighted directed graph $H = (V_H, A_H)$ with $V_H = \{0, 1, \dots, \omega\}$, $A_H = \{(i,j) \in V_H^2 : i < j\}$, and $c_{i,j} \coloneqq W(1+\varepsilon)^{-i}\ell(T_j)$.
3) Compute a shortest $(0,\omega)$-path $P = (n_0, n_1, \dots, n_l)$ with $n_0 = 0$ and $n_l = \omega$ in $H$.
4) For each phase $j \in \{1, \dots, l\}$ explore all unexplored vertices of $V[T_{n_j}]$ in any feasible order using the edge set of $E[T_{n_j}]$.

Let $\sigma_{\text{ALG}}$ be the expanding search pattern given by this algorithm. Let $q \in [0, W]$ be arbitrary and let $j(q) \in \{1, \dots, l\}$ be such that $W - W(1+\varepsilon)^{-n_{j(q)-1}} \leq q < W - W(1+\varepsilon)^{-n_{j(q)}}$. Then we define $\pi(q) \coloneqq \sum_{i=0}^{j(q)} \ell(T_{n_i})$. By $L_q(\sigma_{\text{ALG}})$ we denote the latency of quota $q$ in $\sigma_{\text{ALG}} = (e_1, ..., e_m)$, i.e., the sum of the edge cost of the shortest subsequence $(e_1, ..., e_k)$ of $\sigma_{\text{ALG}}$, such that the tree spanned by $(e_1, ..., e_k)$ has weight at least $q$. The following lemma gives an upper bound on the latency for each quota. Its proof uses the intuitive argument that the worst case for the latency is obtained when all trees are nested and exploration takes place only at the end of each tree. The formal proof can be found in the full version [28].

▶ **Lemma 3.** *The latency of quota $q \in [0, W]$ in $\sigma_{\text{ALG}}$ can be bounded by $L_q(\sigma_{\text{ALG}}) \leq \pi(q)$.*

We can now give an upper bound on $L(\sigma_{\text{ALG}})$. The proof relies on the specific way how the length of the arcs in $H$ are defined. Its proof can be found in the full version [28].

▶ **Lemma 4.** *For the total latency of the algorithm, we have $L(\sigma_{\text{ALG}}) \leq z$ where $z$ is the cost of a shortest $(0,\omega)$-path in $H$.*

The technically most challenging part of the analysis of the algorithm is to bound the cost of a shortest path in relation to the total latency of the optimal expanding search pattern. To this end, we use a probabilistic argument where a distribution over paths in $H$ corresponding to the exploration of trees with exponentially increasing weight is considered.

▶ **Lemma 5.** *Let $\sigma^*$ be an optimal expanding search pattern with total latency $L^* \coloneqq L(\sigma^*)$. Then, a shortest $(0,\omega)$-path in $H$ has cost at most $2(1+\varepsilon)\mathrm{e}L^*$.*

**Proof.** First, we give a lower bound on $L^*$. For this purpose, let $q \in [0, W]$ be arbitrary, and let $\lambda^*(q)$ denote the length of the optimal solution to the instance of the quota version of the rooted prize-collecting Steiner tree problem with quota $q$. Note that there are only finitely many trees $T$ that are subgraphs of $G$ and contain $r$, so $\lambda^*$ is a piece-wise constant function. The optimal expanding search pattern cannot achieve a total weight of $q$ with a latency smaller than $\lambda^*(q)$. Therefore, $L^* \geq \int_0^W \lambda^*(q)\,\mathrm{d}q$.

To show that the cost of the shortest $(0,\omega)$-path in $H$ is bounded from above by $2(1+\varepsilon)\mathrm{e}L^*$, we construct a random path and compute its expected cost. Let $\gamma > 1$ be a parameter whose value will be determined later, and let $b = \gamma^U$ where $U$ is a random variable uniformly drawn from $[0,1)$. We set $\tilde{m} \in \mathbb{N}$ to be the smallest number such that $\lambda^*(W) \leq b\gamma^{\tilde{m}}$. For $j \in \{0,\dots,\tilde{m}\}$, let $\tilde{n}_j \coloneqq \max\{k \in \{0,\dots,\omega\} : \ell(T_k) \leq 2b\gamma^j\}$. These values are well-defined as $\ell(T_0) = 0$. Note that the sequence $\tilde{n}_0, \tilde{n}_1, \dots, \tilde{n}_{\tilde{m}}$ is non-decreasing. We denote by $n_0, n_1, \dots, n_m$ the longest increasing subsequence of $\tilde{n}_0, \tilde{n}_1, \dots, \tilde{n}_{\tilde{m}}$. In the following, we compute the cost of the path $P = (n_0, n_1, \dots, n_m)$. Let $i \in \{0,\dots,m\}$ be such that $W - W(1+\varepsilon)^{-i} \leq q < W - W(1+\varepsilon)^{-(i+1)}$. Recall that $\pi(q) = \sum_{j=0}^{i+1} \ell(T_{n_j})$ is an upper bound on the latency of quota $q$, i.e., the sum of the lengths of all trees that lie on path $P$ up to the first tree that collects a quota of size $q$ all by itself.

For a quota $q \in [0,W]$ we find it convenient to denote by $\bar{q} \coloneqq W - q$ the quota left aside. Let $\bar{q} \in [W(1+\varepsilon)^{-\omega}, W]$ be arbitrary, and let $j \in \{0,\dots,m\}$ and $d \in [1,\gamma)$ be such that $\lambda^*(W-\bar{q}) = d\gamma^j$. We distinguish two cases regarding the relation between $b$ and $d$.

**First case: $d \leq b$.** Since $\lambda^*(q) = \lambda^*(W-\bar{q}) = d\gamma^j$, there is a tree of cost $d\gamma^j$ that contains the root and explores a total weight of at least $W - \bar{q}$. When computing the 2-approximation for the quota version of the prize-collecting Steiner tree problem with quota $W - W(1+\varepsilon)^{-i}$, we obtain a tree $T_i$ with length $\ell(T_i) \leq 2\lambda^*\big(W - W(1+\varepsilon)^{-i}\big) \leq 2\lambda^*(W-\bar{q}) = 2d\gamma^j$. The first inequality is obtained by using the 2-approximation and the second inequality from $\lambda^*$ being non-decreasing. Since $\ell(T_i) \leq 2d\gamma^j \leq 2b\gamma^j$, we have that $n_j \geq i$. Using that $\pi(q)$ is non-decreasing, that $W - W(1+\varepsilon)^{-i} \geq W - (1+\varepsilon)\bar{q}$, and that $\gamma > 1$, we obtain

$$\pi\big(W - (1+\varepsilon)\bar{q}\big) \leq \sum_{k=0}^{j} \ell(T_{n_k}) \leq \sum_{k=0}^{j} 2b\gamma^k = 2b\frac{\gamma^{j+1}-1}{\gamma-1} \leq 2b\gamma^j\frac{\gamma}{\gamma-1}.$$

**Second case: $d > b$.** Analogously to the first case we obtain $\ell(T_i) \leq 2d\gamma^j$. However, with $1 \leq b$ and $d < \gamma$ we have $d < b\gamma$ which yields $\ell(T_i) \leq 2d\gamma^j < 2b\gamma^{j+1}$. Hence, we have that $n_{j+1} \geq i$. Again, using that $\pi(q)$ is non-decreasing, that $W - W(1+\varepsilon)^{-i} \geq W - (1+\varepsilon)\bar{q}$, and that $\gamma > 1$, we obtain

$$\pi\big(W - (1+\varepsilon)\bar{q}\big) \leq \sum_{k=0}^{j+1} \ell(T_{n_k}) \leq \sum_{k=0}^{j+1} 2b\gamma^k = 2b\frac{\gamma^{j+2}-1}{\gamma-1} \leq 2b\gamma^{j+1}\frac{\gamma}{\gamma-1}.$$

Note that we are in the first case when $U \in [\log_\gamma d, 1]$ and in the second case when $U \in [0, \log_\gamma d)$. Taking the expectation over $U$, we obtain

$$\begin{aligned}
\mathbb{E}_U\big[\pi\big(W-(1+\varepsilon)\bar{q}\big)\big] &\leq \int_{\log_\gamma d}^{1} 2b\gamma^j\frac{\gamma}{\gamma-1}\,\mathrm{d}U + \int_{0}^{\log_\gamma d} 2b\gamma^{j+1}\frac{\gamma}{\gamma-1}\,\mathrm{d}U \\
&= 2\gamma^j\frac{\gamma}{\gamma-1}\left[\int_{\log_\gamma d}^{1}\gamma^U\,\mathrm{d}U + \gamma\int_{0}^{\log_\gamma d}\gamma^U\,\mathrm{d}U\right] = 2\gamma^j\frac{\gamma}{\gamma-1}\left[\frac{\gamma-d}{\ln\gamma} + \gamma\frac{d-1}{\ln\gamma}\right] \\
&= 2\gamma^j d\frac{\gamma}{\ln\gamma} = 2\frac{\gamma}{\ln\gamma}\lambda^*(W-\bar{q}).
\end{aligned}$$

Next, consider the case that $\bar{q} < W(1+\varepsilon)^{-\omega}$ is arbitrary. Let $j \in \{0,\dots,m\}$ and $d \in [1,\gamma)$ be such that $\lambda^*(W-\bar{q}) = d\gamma^j$. By the choice of $\omega$, we have $W - \bar{q} > W - W(1+\epsilon)^{-\omega} > W - 1$, and, hence, $\lambda^*(W) = \lambda^*(W-\bar{q})$. We distinguish two cases regarding the relation of $b$ and $d$.

**First case: $d \leq b$.** Since $\lambda^*(W) = d\gamma^j$, there is a tree of cost $d\gamma^j$ containing the root that explores a total weight of at least $W - \bar{q}$. When computing the 2-approximation for the quota version of the prize-collecting Steiner tree problem with quota $W - W(1+\varepsilon)^{-\omega}$, we obtain a tree $T_\omega$ with length $\ell(T_\omega) \leq 2\lambda^*(W) = 2d\gamma^j \leq 2b\gamma^j$, which yields $n_j \geq \omega$, and thus, $j = m$. By using $\gamma > 1$, we obtain

$$\pi\big(W\big) \leq \sum_{k=0}^{m} \ell(T_{n_k}) \leq \sum_{k=0}^{m} 2b\gamma^k = 2b\frac{\gamma^{m+1}-1}{\gamma-1} \leq 2b\gamma^m \frac{\gamma}{\gamma-1}.$$

**Second case: $d > b$.** Analogously to the first case, we obtain $\ell(T_\omega) \leq 2d\gamma^j$. However, with $1 \leq b$ and $d < \gamma$ we have $d < b\gamma$ which yields $\ell(T_\omega) \leq 2d\gamma^j < 2b\gamma^{j+1}$. Hence, we have that $n_{j+1} \geq \omega$, i.e., $j \geq m-1$. In any case, by using that $\gamma > 1$, we obtain

$$\pi(W) \leq \sum_{k=0}^{m} \ell(T_{n_k}) \leq \sum_{k=0}^{m} 2b\gamma^k = 2b\frac{\gamma^{m+1}-1}{\gamma-1} \leq 2b\gamma^m \frac{\gamma}{\gamma-1}.$$

Again, we are in the first case when $U \in [\log_\gamma d, 1]$ and in the second case when $U \in [0, \log_\gamma d)$. Taking the expectation over $U$, we obtain

$$\begin{aligned}
\mathbb{E}_U\big[\pi\big(W\big)\big] &\leq \int_{\log_\gamma d}^{1} 2b\gamma^j \frac{\gamma}{\gamma-1}\,\mathrm{d}U + \int_{0}^{\log_\gamma d} 2b\gamma^{j+1}\frac{\gamma}{\gamma-1}\,\mathrm{d}U \\
&= 2\gamma^j \frac{\gamma}{\gamma-1}\left[\int_{\log_\gamma d}^{1} \gamma^U\,\mathrm{d}U + \gamma\int_{0}^{\log_\gamma d}\gamma^U\,\mathrm{d}U\right] = 2\gamma^j\frac{\gamma}{\gamma-1}\left[\frac{\gamma-d}{\ln\gamma} + \gamma\frac{d-1}{\ln\gamma}\right] \\
&= 2\gamma^j d\frac{\gamma}{\ln\gamma} = 2\frac{\gamma}{\ln\gamma}\lambda^*(W). \qquad (1)
\end{aligned}$$

The expected cost of the $(0,\omega)$-path $P = (n_0, n_1, \ldots, n_m)$ is then given by

$$\mathbb{E}\big[c(P)\big] = \mathbb{E}\left[\int_0^W \pi(q)\,\mathrm{d}q\right] = \mathbb{E}\left[\int_0^W \pi(W-\bar{q})\,\mathrm{d}\bar{q}\right].$$

As $\pi(q)$ is piece-wise constant, we exchange the order of expectation and integral such that

$$\begin{aligned}
\mathbb{E}\big[c(P)\big] &= \int_0^W \mathbb{E}\big[\pi(W-\bar{q})\big]\,\mathrm{d}\bar{q} \\
&= -(1+\varepsilon)\left(\int_{W(1+\varepsilon)^{-1}}^{W(1+\varepsilon)^{-\omega}} \mathbb{E}\big[\pi\big(W-(1+\varepsilon)\bar{q}\big)\big]\,\mathrm{d}\bar{q} + \int_{W(1+\varepsilon)^{-\omega}}^{0} \mathbb{E}\big[\pi\big(W-(1+\varepsilon)\bar{q}\big)\big]\,\mathrm{d}\bar{q}\right) \\
&\leq (1+\varepsilon)\int_{W(1+\varepsilon)^{-\omega}}^{W(1+\varepsilon)^{-1}} \mathbb{E}\big[\pi\big(W-(1+\varepsilon)\bar{q}\big)\big]\,\mathrm{d}\bar{q} + W(1+\varepsilon)^{-(\omega-1)}\mathbb{E}\big[\pi(W)\big],
\end{aligned}$$

where we further used the substitution rule for integrals and the fact that $\pi$ is non-decreasing. Using (1), we further obtain

$$\begin{aligned}
\mathbb{E}\big[c(P)\big] &\leq \frac{2\gamma(1+\varepsilon)}{\ln\gamma}\left[\int_{W(1+\varepsilon)^{-\omega}}^{W(1+\varepsilon)^{-1}} \lambda^*(W-\bar{q})\,\mathrm{d}\bar{q} + W(1+\varepsilon)^{-\omega}\mathbb{E}\big[\lambda^*(W)\big]\right] \\
&= \frac{2\gamma(1+\varepsilon)}{\ln\gamma}\int_0^{W(1+\varepsilon)^{-1}} \lambda^*(W-\bar{q})\,\mathrm{d}\bar{q},
\end{aligned}$$

where for the equation we used that $W(1+\varepsilon)^{-\omega} < 1$ and, hence, $\lambda^*$ is constant on the interval $[W - W(1+\varepsilon)^{-\omega}, W]$. Finally, we obtain

$$\mathbb{E}\big[c(P)\big] \leq \frac{2\gamma(1+\varepsilon)}{\ln\gamma}\int_0^W \lambda^*(W-\bar{q})\,\mathrm{d}\bar{q} = \frac{2\gamma(1+\varepsilon)}{\ln\gamma}\int_0^W \lambda^*(q)\,\mathrm{d}q \leq \frac{2\gamma(1+\varepsilon)}{\ln\gamma}L^*.$$

This shows that, in the expectation over $U$, $P$ has an expected cost of at most $\frac{2\gamma(1+\varepsilon)}{\ln\gamma}L^*$. Therefore, we obtain the same bound on the cost of the shortest path. This term is minimized for $\gamma = \mathrm{e}$, which implies the result. ◀

Theorem 1 now follows from combining Lemma 4 and Lemma 5.

## 4 The unweighted case

Compared to the weighted case we do the following adjustments in order to obtain a 2e-approximation. First, instead of using the quota problem of $k$-MST, for all $k \in \{1,\dots,n\}$ we solve the original $k$-MST problem with the 2-approximation algorithm of Garg [25] and obtain $n$ trees $T_1,\dots,T_n$. The auxiliary weighted directed graph $H = (V_H, A_H)$ is then defined by $V_H = \{1,2,\dots,n\}$, $A_H = \{(i,j) \in V_H^2 : i < j\}$, and $\ell_{i,j} := (n-i)c(T_j)$. Finally, we compute a shortest $(1,n)$-path $P = (n_0, n_1, \dots, n_l)$ with $n_0 = 1$ and $n_l = n$ in $H$. For each phase $j \in \{1,\dots,l\}$, we explore all unexplored vertices of $V[T_{n_j}]$ in any feasible order using the edge-set of $E[T_{n_j}]$. The better approximation factor is due to the fact that we save the factor of $(1+\varepsilon)$ since we can compute the $k$-MSTs for all relevant values of $k$, whereas before we needed a rounding technique. We then obtain the following result, which we prove in the full version [28].

▶ **Theorem 6.** *There is a polynomial-time* 2e*-approximation algorithm for the unweighted expanding search problem.*

## 5 The Euclidean case

In this section, we show the following theorem.

▶ **Theorem 7.** *On Euclidean graphs, there exists a PTAS for ESP.*

Our approach has three steps, corresponding to the three subsections of this section. The first two steps are reductions inspired by Sitters [40]. In the first step, we show a reduction from ESP to a problem called $\delta$-bounded ESP, for some constant $\delta \in \mathbb{R}_+$, in the sense that a PTAS for $\delta$-bounded ESP implies a PTAS for ESP. In the next step, we reduce the latter problem to another problem called $\kappa$-segmented ESP, for some constant $\kappa \in \mathbb{N}$, with weights in $\{0,1\}$, in the same sense as before. Finally, we provide a PTAS for the latter problem in the Euclidean case using ideas by Arora [9] as well as Sitters [40].

We define the auxiliary subproblems as modifications of ESP. First, in $\delta$-bounded ESP, the input comes with an additional delay parameter $D \geq 0$, and it is guaranteed that there exists a solution visiting all nonzero-weight vertices and completing by time $\delta D$, i.e. this solution has length $\delta D$ (recall definition in Section 2). The objective is minimizing $L'(\sigma) = \sum_{v\in V^*} w_v L'_v(\sigma)$ where $L'_v(\sigma) = D + L_v(\sigma)$. Second, in $\kappa$-segmented ESP, the output needs to come with $\kappa+1$ additional numbers $0 = t^{(0)} \leq t^{(1)} \leq \cdots \leq t^{(\kappa)}$. For $v \in V$, its rounded search time is then $\bar{L}_v(\sigma) = \inf\{t^{(i)} : 0 \leq i \leq \kappa,\, L_v(\sigma) \leq t^{(i)}\}$, and the objective is minimizing $\bar{L}(\sigma) = \sum_{v\in V^*} w_v \bar{L}_v(\sigma)$.

We assume $0 < \varepsilon \leq 1$ and use $O_\varepsilon(f)$ to denote $O(f)$ when $\varepsilon$ is a constant.

## 5.1 Reducing ESP to $\delta$-Bounded ESP

In this subsection, we show the following lemma.

▶ **Lemma 8.** *Consider any class $\mathcal{C}$ of metric spaces and constants $\alpha > 1, \varepsilon > 0$. There exists a constant $\delta$ such that, if there exists a polynomial-time $\alpha$-approximation algorithm for $\delta$-bounded ESP on $\mathcal{C}$, then there exists an $(\alpha + \varepsilon)$-approximation algorithm for ESP on $\mathcal{C}$.*

We follow the decomposition approach by Sitters [40] and adapt it to ESP at several places. To do so, we assume that a polynomial-time $\alpha$-approximation algorithm for $\delta$-bounded ESP on $\mathcal{C}$, denoted $\textsc{Approx}_{\delta\text{-bd}}$ in the following, is given for a yet-to-be-determined value of $\delta$. In the remainder of this subsection we describe, given any $\varepsilon > 0$, a polynomial-time algorithm for ESP on $\mathcal{C}$ based on this, and we show that it is a $(\alpha + O(\varepsilon))$-approximation algorithm.

For some constant $\beta$, we need, in addition to $\textsc{Approx}_{\delta\text{-bd}}$, a polynomial-time $\beta$-approximation algorithm $\textsc{Approx}_\beta$ for ESP on $\mathcal{C}$ as a subroutine. We emphasize that *any* constant $\beta$ is sufficient to obtain an approximation guarantee of $\alpha + \varepsilon$ in polynomial time. Therefore, we can pick, e.g., the algorithm from Section 3. For notational purposes, we assume $\alpha \neq \beta$.

In our algorithm, we apply $\textsc{Approx}_\beta$ to obtain an order of the vertices according to their search times in the solution, and we obtain a partition of the vertices by cutting this order at several places. We run $\textsc{Approx}_{\delta\text{-bd}}$ on the (carefully defined) emerging subinstances. We can, however, not simply concatenate all these solutions because any of these solutions may, despite its low cost, have large total length, which would delay the solutions of all later subinstances. We solve this issue by cutting the solution at a certain point and using the solution given by $\textsc{Approx}_\beta$ from then on – a solution with a length bound.

In the following, we present our algorithm which is given some $\varepsilon > 0$ as well as an instance $I$ of ESP on $\mathcal{C}$. Our algorithm has five steps.

1) **Approximate:** Apply $\textsc{Approx}_\beta$ to the instance to obtain a solution $\sigma_\beta$.
2) **Partition:**
   - Define $\gamma := 3/\varepsilon$, $a := \beta\gamma/\varepsilon$, and pick $b$ uniformly at random in $[0, a]$.
   - Define time points $t_i := \mathrm{e}^{(i-2)a+b}$ for $i \in [q+1]$, where $q$ is as small as possible such that $L_v(\sigma_\beta) < t_{q+1}$ for all $v \in V$.
   - For $i \in [q]$, let $V_i := \{v \in V : t_i \leq L_v(\sigma_\beta) < t_{i+1}\}$ and $U_i := V_1 \cup \cdots \cup V_i$.
   - For $i \in [q]$, define $I_i$ to be an instance which is obtained from $I$ by setting the weight of all vertices in $V \setminus V_i$ to zero. Note that $I_i$ with delay parameter $\gamma t_i$ is an instance of $(\mathrm{e}^a/\gamma)$-bounded ESP. Indeed, the prefix $\sigma_{\beta,i+1}$ of $\sigma_\beta$ visiting $U_{i+1}$ has total length at most $(\mathrm{e}^a/\gamma)\gamma t_i = t_{i+1}$.
3) **Approximate subproblems:** For $i \in [q]$, apply $\textsc{Approx}_{\delta\text{-bd}}$ to $I_i$ to obtain an $\alpha$-approximation $\sigma_{\alpha,i}$.
4) **Modify:** For each $i \in [q]$, define $\sigma_i$ to be $\sigma'_{\alpha,i} + \sigma_{\beta,i+1}$ where:
   - $\sigma'_{\alpha,i}$ is the longest prefix of $\sigma_{\alpha,i}$ of length at most $(1 + \mathrm{e}^a/\varepsilon\gamma)\gamma t_i$.
   - $\sigma_{\beta,i+1}$ is the prefix of $\sigma_\beta$ visiting $U_{i+1}$.
5) **Concatenate:** Return $\sigma_1 + \cdots + \sigma_q$.

We show Lemma 8 by establishing two lemmata on the above algorithm. We first prove that partitioning the instance into multiple instances of $(\mathrm{e}^a/\gamma)$-bounded ESP is only at the loss of a $1 + \varepsilon$ factor in the achievable (total) objective-function value. Formally, we denote by $\sigma^*$ an optimal solution for $I$ and, for all $i \in [q]$, by $\sigma_i^*$ an optimal solution for $I_i$. The proofs of the following two lemmata can be found in the full version [28].

▶ **Lemma 9.** *It holds that* $\mathbb{E}\left[\sum_{i\in[q]} L'(\sigma_i^*)\right] \leq (1+\varepsilon)L(\sigma^*)$.

The next lemma is concerned with Step 4 of the algorithm. For all $i \in [q]$, it bounds the cost of $\sigma_i$ against the cost of $\sigma_i^*$, and it also bounds the total length of $\sigma_i$.

▶ **Lemma 10.** *For each $i \in [q]$, the total length of $\sigma_i$ is at most $\gamma t_{i+1} - \gamma t_i$. Furthermore, it holds that $L'(\sigma_i) \leq \alpha(1+\varepsilon)L'(\sigma_i^*)$.*

With these lemmata at hand, Lemma 8 easily follows.

**Proof of Lemma 8.** Note that Lemma 10 implies that, in the concatenation of $\sigma_1, \ldots, \sigma_q$, $\sigma_i$ starts after a total length of at most $\gamma t_i$, for all $i \in [q]$. Therefore, again by Lemma 10, the cost of the concatenation, as a solution to $I$, has expected cost at most the left-hand side of the inequality in Lemma 10 summed over all $i \in [q]$. Hence, applying this inequality, taking expectation, and then applying Lemma 9 completes the proof. ◀

The partition as stated in the decomposition algorithm is a random partition. We note that the algorithm can be derandomized using the same techniques as in [40], i.e., by enumerating all partitions.

We also observe that, from now on, we may also assume that all weights are in $\{0, 1\}$. This is due to the following lemma, proven by Sitters [40] for pathwise search, but the same proof works in our case as shown in the full version [28].

▶ **Lemma 11** (See [40], Lemma 2.10)**.** *Consider any class $\mathcal{C}$ of metric spaces and any constants $\alpha > 0, \delta, \varepsilon > 0$. If there exists a polynomial-time $\alpha$-approximation algorithm for $\delta$-bounded ESP with weights in $\{0, 1\}$, then there exists a polynomial-time $(\alpha+\varepsilon)$-approximation algorithm for $\delta$-bounded ESP.*

Note that Lemma 10 and 11, together with Theorem 6, yields an alternative polynomial-time $(2\mathrm{e} + \varepsilon)$-approximation algorithm for the general ESP (Theorem 1).

## 5.2 Reducing $\delta$-Bounded ESP to $\kappa$-Segmented ESP

The following lemma can be proven analogously to a lemma of Sitters [40].

▶ **Lemma 12** (See [40], Lemma 2.14)**.** *Consider any class of metric spaces $\mathcal{C}$, any class of weights $\mathcal{W}$, and any constants $\alpha > 1, \delta, \varepsilon > 0$. If, for each constant $\kappa$, there exists a polynomial-time $\alpha$-approximation algorithm for $\kappa$-segmented ESP on $\mathcal{C}$ with weights $\mathcal{W}$, then there exists a polynomial-time $(\alpha + \varepsilon)$-approximation algorithm for $\delta$-bounded ESP $\mathcal{C}$ with weights $\mathcal{W}$.*

In the proof of the lemma, a similar idea as for the proof of Lemma 10 is used to show that there is a cheap solution that completes before time $O_\varepsilon(1+\delta)D$, where $D$ is the given delay of the instance. Then, by considering appropriate time points starting at $D$ and growing exponentially with base $(1 + \Theta(\varepsilon))$, one can show that for $\kappa \in O_\varepsilon(\log(1+\delta))$, an $\alpha$-approximate solution for the $\kappa$-segmented version of the instance can be transformed into the desired $((1+\varepsilon)\alpha)$-approximate solution for the original instance.

## 5.3 A PTAS for $\kappa$-Segmented ESP in the Euclidean Case

Sitters [40] observed that, in Euclidean space, the QPTAS for the traveling repairperson problem [10] (which is based on the well-known PTAS by Arora for TSP [9]) can be turned into a PTAS for the segmented version of the traveling repairperson problem. In this section, we observe that an adapted approach yields a PTAS for Euclidean segmented ESP with

weights in $\{0, 1\}$. We focus on the two-dimensional case; an extension to the $d$-dimensional case for constant $d$ is straightforward. The following description is self-contained up to parts deferred to the full version [28], but familiarity with Arora's PTAS [9] may still be helpful.

### 5.3.1 Setup

The core of our PTAS for segmented ESP is the dynamic-programming procedure. Before this procedure is called, however, there are several preprocessing steps. First, consider a smallest axis-aligned square that contains all weight-1 vertices from the input. We denote it by $S_0$ and its side length by $d_0$. Note that $d_0$ is a lower bound on the cost of an optimal solution. An optimal solution is, however, not necessarily entirely contained in the square as it may use a weight-0 vertex outside the square as a "Steiner" vertex. We therefore enlarge the square from its center by a factor of $3n^2 + 1$, yielding a new square $S$ with side length $d = (3n^2 + 1)d_0$. The scaling factor is chosen such that all points whose distance from $S_0$ is at least $\sqrt{2}n^2 d_0$ are included. Note that there exists an optimal solution that is entirely contained in $S$ because a trivial upper bound on the cost of the optimal solution of $\sqrt{2}n^2 d_0$ can be obtained by connecting all weight-1 vertices to $r$. We can therefore ignore all input points outside $S$.

**Round the instance.** We place a grid of granularity $\Theta(\varepsilon d/n^4)$ within $S$ and move each input point to a closest grid point. Note that, this way, several input points may end up at the same location. In the same way as in the literature [10], any solution for the rounded instance can be turned into a solution for the original instance at a cost of $O(\varepsilon)\mathrm{OPT}$ in the objective-function value: The additional cost of $O(\varepsilon d/n^3)$ per vertex can be charged to the objective as it is $\Omega(d/n^2)$ by construction of $S$.

**Build random quadtree.** We first obtain an even larger square from $S$ by enlarging it by an additional factor of 2 from its center and then shifting it to the left by a value $a$ chosen uniformly at random from $\{-d/2, -d/2 + 1, \ldots, d/2 - 1, d/2\}$ and to the top by a value $b$ chosen uniformly at random from the same set, independently from $a$. Note that, in any event, the resulting square $S'$ contains $S$.

We partition $S'$ into four equal-sized squares, which are recursively partitioned in the same way until they only contain a single grid point at which there is a vertex (but possibly many vertices). From this partition, a so-called quadtree naturally emerges by identifying each of the squares (also called cells in the following) with a node and making a node a child of another node if its corresponding square is one of the four smaller squares within that node's square. We root the quadtree at the node corresponding to $S'$. Since the minimum distance between any two vertices not at the same grid point is $\Theta(\varepsilon d/n^4)$ by the rounding step, the quadtree has depth $O(\log d)$.

**Derandomization.** We remark that the randomization is only for a simpler analysis. Indeed, our algorithm can be derandomized in the same way as Arora's PTAS and its variants: Simply try all, polynomially many, values for the random variables $a$ and $b$, and output the cheapest solution obtained this way.

### 5.3.2 Portal-respecting solutions and the structural result

The set of solutions over which the dynamic-programming procedure optimizes are so-called portal-respecting solutions. Such solutions only cross cell boundaries at so-called portals, and they do so only a constant number of times at each portal. For each cell, we place

$\Theta(\log n/\varepsilon)$ equidistant portals on each side of that cell, from corner to corner and including the corners. Additionally, each cell inherits all portals from all its ancestors in the quadtree. The following structural result states that we only lose a $1 + O(\varepsilon)$ factor when restricting to portal-respecting solutions.

▶ **Lemma 13.** *With constant probability (over the random placement of the quadtree), there exists a $(1 + O(\varepsilon))$-approximate portal-respecting solution.*

The result can be proved precisely in the same way as in [10], by applying Arora's structural result [9] to each segment. In [10], the pathwise version of our problem is considered, but this difference does not affect the proof.

### 5.3.3 Further setup

Before we describe the dynamic program, we need two additional setup steps.

**Additional rounding.** Since we guess lengths of parts of the solution, we assume at the loss of another $1+O(\varepsilon)$ factor that the distance between any two relevant points (i.e., input points or portals) is a polynomially bounded integer. This is possible because the objective-function value is the sum of polynomially many distances.

**Guessing of segment lengths.** It will be useful to know the completion times $t^{(1)}, \dots, t^{(\kappa)}$ before running the dynamic-programming procedure. By our rounding procedure, we know that there are only $n^{O(1)}$ options for each of these $O(1)$ lengths, meaning that there are $n^{O(1)}$ combinations of different completion times for each of the segments, which we can all guess.

### 5.3.4 Dynamic programming

For each cell $z$ of the quadtree we additionally "guess" the following pieces of information relevant for the other quadtree cells (reflected in the fact that there is a DP entry for each combination). Specifically, for each segment $i \in [\kappa]$, we guess

- **(i)** the total length $\ell_i$ of segment $i$ within the cell,
- **(ii)** the number $m_i$ of times that the segment crosses the boundary of the cell, and for each $j \in [m_i]$ of these crossing a *type* $\tau_{i,j}$ for the $j$-th such time, containing
  - the portal $p_{i,j}$ at which the cell is intersected, and
  - whether the segment enters or leaves the cell at $p_{i,j}$.

Note that, again, there are only polynomially many options for each of the parameters (in particular, $m_i$ can be assumed to be at most $O(\log n/\varepsilon)$, and we only have constantly many options for the type of each crossing) and therefore only polynomially many DP entries.

Any DP entry $\mathrm{DP}[z, (\ell_i, (\tau_{i,j})_{j\in[m_i]})_{i\in[\kappa]}]$ is supposed to contain the cost of the cheapest portal-respecting solution restricted to the corresponding cell obeying the constraints imposed by the guessed parameters and visiting all vertices within the cell. Note that such a solution may not exist (e.g., the cell does not contain the root but some other vertices, and no segment ever enters the cell), in which case the cost is $\infty$. Otherwise, the cost of a solution restricted to a cell refers to the sum over all vertices in that cell of the completion time of the segment that they are visited in.

With this definition, the entry $\mathrm{DP}[z_0, (t^{(i)} - \sum_{i'<i} t^{(i')}, ())_{i\in[\kappa]}]$ is supposed to contain the cost of the optimal portal-respecting solution, where $z_0$ is the root of the quadtree and $()$ is the empty tuple. By standard techniques, the actual solution can be recovered from these entries. The DP entries can be computed in a fairly standard manner. A more detailed description can be found in the full version [28].

## 6 Hardness of approximation

This section is dedicated to the following theorem.

▶ **Theorem 14.** *There exists some constant $\varepsilon > 0$ such that there is no polynomial-time $(1+\varepsilon)$-approximation algorithm for the expanding search problem, unless* P = NP.

The hardness result for ESP follows by a reduction from a variant of the Steiner tree problem which is defined as follows. Given a graph $G = (V, E)$ with non-negative edge costs and a set $T \subseteq V$ of vertices, the so-called terminals, the Steiner tree problem on graphs asks for a minimum-cost tree that is a subgraph of $G$ and that contains all vertices in $T$. The variant that we consider and use is the so-called SteinerTree(1,2), short ST(1,2), where $G$ is a complete graph and all edge costs are either 1 or 2. Bern and Plassmann [14] showed the following theorem.

▶ **Theorem 15** (Theorem 4.2 in [14]). *SteinerTree(1,2) is* MaxSNP*-hard.*

It was shown in [11] that there exists no polynomial-time approximation scheme for any MaxSNP-hard problem, unless P = NP. Hence, there exists some constant $\rho > 0$ such that there is no polynomial-time $(1+\rho)$-approximation algorithm for ST(1,2), unless P = NP. We use this to show the hardness result for ESP.

The main idea of the proof of Theorem 14 is as follows. Given a $\beta$-approximation algorithm Alg$'$ for the expanding search problem for any $\beta > 1$, we construct a $\gamma$-approximation algorithm Alg for ST(1,2) with $\gamma < 1 + \rho$. With the approximation hardness of ST(1,2), this contradicts the existence of a $\beta$-approximation algorithm Alg$'$ for the expanding search problem for any $\beta > 1$. The construction of the ESP instance in the reduction from ST(1,2) is similar to the one used for the travelling repairperson problem. Therein, we construct several copies of the ST(1,2) instance, which are then connected to a root vertex with an edge of high cost. However, a significant challenge is that we need to prove that the obtained expanding search sequence fulfills a property which we call *structured*. Intuitively, this property means that no copy of the original ST(1,2) instance is visited more than once and that all edges belonging to one copy are contiguous within the expanding search pattern. This property is trivial for the travelling repairperson problem since here using an expensive edge from the root to one of the copies more than once increases the total cost significantly. In the expanding search problem, however, these costs are not paid multiple times. The exact construction of the ESP instance and the proof of Theorem 14 can be found in the full version [28].

## References


1 Foto N. Afrati, Stavros S. Cosmadakis, Christos H. Papadimitriou, George Papageorgiou, and Nadia Papakostantinou. The complexity of the travelling repairman problem. *RAIRO – Theoretical Informatics and Applications*, 20(1):79–87, 1986.

2 Steve Alpern, Robbert Fokkink, Leszek Gasieniec, Roy Lindelauf, and V. S. Subrahmanian. *Search Theory*. Springer, New York, 2013.

3 Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*, volume 55 of *International Series in Operations Research and Management Science*. Kluwer, 2003.

4 Steve Alpern and Thomas Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61(2):265–279, 2013. doi:10.1287/opre.1120.1134.

5 Steve Alpern and Thomas Lidbetter. Approximate solutions for expanding search games on general networks. *Annals of Opererations Research*, 275(2):259–279, 2019.

6 Spyros Angelopoulos, Christoph Dürr, and Thomas Lidbetter. The expanding search ratio of a graph. *Discrete Applied Mathematics*, 260:51–65, 2019.

7 Aaron Archer and Anna Blasiak. Improved approximation algorithms for the minimum latency problem via prize-collecting strolls. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 429–447, 2010.

8 Aaron Archer, Asaf Levin, and David P. Williamson. A faster, better approximation algorithm for the minimum latency problem. *SIAM Journal on Computing*, 37(5):1472–1498, 2008.

9 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

10 Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM Journal on Computing*, 32(5):1317–1337, 2003.

11 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

12 Igor Averbakh. Emergency path restoration problems. *Discrete Optimization*, 9(1):58–64, 2012.

13 Igor Averbakh and Jordi Pereira. The flowtime network construction problem. *IIE Transactions*, 44(8):681–694, 2012.

14 Marshall Bern and Paul Plassmann. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32(4):171–176, 1989.

15 Avrim Blum, Prasad Chalasani, Don Coppersmith, William R. Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–171, 1994.

16 Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 36–45, 2003.

17 Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 72–83, 2004.

18 Thijs Dewilde, Dirk Cattrysse, Sofie Coene, Frits C. R. Spieksma, and Pieter Vansteenwegen. Heuristics for the traveling repairman problem with profits. *Computers & Operations Research*, 40:1700–1707, 2013.

19 Jittat Fakcharoenphol, Chris Harrelson, and Satish Rao. The $k$-traveling repairmen problem. *ACM Transactions on Algorithms*, 3(4):40:1–40:16, 2007.

20 Matteo Fischetti, Gilbert Laporte, and Silvano Martello. The delivery man problem and cumulative matroids. *Operations Research*, 41:1010–1176, 1993.

21 Zachary Friggstad, Mohammad R. Salavatipour, and Zoya Svitkina. Asymmetric traveling salesman path and directed latency problems. *SIAM Journal on Computing*, 42:1596–1619, 2013.

22 Shmuel Gal. Search games with mobile and immobile hider. *SIAM Journal on Control and Optimization*, 17:99–122, 1979.

23 Shmuel Gal. *Search Games*. Academic Press, New York, 1980.

24 Alfredo García, Pedro Jodrá, and Javier Tejel. A note on the travelling repairman problem. *Networks*, 40:27–31, 2002.

25 Naveen Garg. Saving an epsilon: A 2-approximation for the $k$-MST problem in graphs. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 396–402, 2005.

26 Michel X. Goemans and Jon M. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82:111–124, 1998.

27 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

28 Svenja M Griesbach, Felix Hommelsheim, Max Klimm, and Kevin Schewior. Improved approximation algorithms for the expanding search problem. *arXiv preprint arXiv:2301.03638*, 2023.

29 Ben Hermans, Roel Leus, and Jannik Matuschke. Exact and approximation algorithms for the expanding search problem. *INFORMS Journal on Computing*, 34(1):281–296, 2022.

30 Rufus Isaacs. *Differential Games*. John Wiley and Sons, New York, 1965.

31 David S. Johnson, Maria Minkoff, and Steven Philipps. The prize collecting Steiner tree problem: Theory and practice. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 760–769, 2000.

32 David Kirkpatrick. Hyperbolic dovetailing. In *Proceedings of the Annual European Symposium on Algorithms (ESA)*, pages 516–527, 2009.

33 Elias Koutsoupias, Christos H. Papadimitriou, and Mihalis Yannakakis. Searching a fixed graph. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 280–289, 1996.

34 Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, and Leen Stougie. News from the online traveling repairman. *Theoretical Computer Science*, 295:279–294, 2003.

35 Songtao Li and Simin Huang. Multiple searchers searching for a randomly distributed immobile target on a unit network. *Networks*, 71(1):60–80, 2018.

36 Edward Minieka. The delivery man problem on a tree network. *Annals of Operations Research*, 18:261–266, 1989.

37 Viswanath Nagarajan and R. Ravi. The directed minimum latency problem. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 193–206. Springer, 2008.

38 Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23:555–565, 1976.

39 René Sitters. The minimum latency problem is NP-hard for weighted trees. In *Proceedings of the International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 230–239, 2002.

40 René Sitters. Polynomial time approximation schemes for the traveling repairman and other minimum latency problems. *SIAM Journal on Computing*, 50(5):1580–1602, 2021.

41 Yushi Tan, Feng Qiu, Arindam K. Das, Daniel S. Kirschen, Payman Arabshahi, and Jianhui Wang. Scheduling post-disaster repairs in electricity distribution networks. *IEEE Trans. Power Syst.*, 34(4):2611–2621, 2019.

42 K. E. Trummel and J. R. Weisinger. Technical note – The complexity of the optimal searcher path problem. *Opererations Research*, 34(2):324–327, 1986.

43 John N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:262–283, 1992.

# Noisy k-Means++ Revisited

**Christoph Grunau**
ETH Zürich, Switzerland

**Ahmet Alper Özüdoğru**
ETH Zürich, Switzerland

**Václav Rozhoň**
ETH Zürich, Switzerland

## Abstract

The $k$-means++ algorithm by Arthur and Vassilvitskii [SODA 2007] is a classical and time-tested algorithm for the $k$-means problem. While being very practical, the algorithm also has good theoretical guarantees: its solution is $O(\log k)$-approximate, in expectation.

In a recent work, Bhattacharya, Eube, Roglin, and Schmidt [ESA 2020] considered the following question: does the algorithm retain its guarantees if we allow for a slight adversarial noise in the sampling probability distributions used by the algorithm? This is motivated e.g. by the fact that computations with real numbers in $k$-means++ implementations are inexact. Surprisingly, the analysis under this scenario gets substantially more difficult and the authors were able to prove only a weaker approximation guarantee of $O(\log^2 k)$. In this paper, we close the gap by providing a tight, $O(\log k)$-approximate guarantee for the $k$-means++ algorithm with noise.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Theory of computation → Unsupervised learning and clustering

**Keywords and phrases** clustering, k-means, k-means++, adversarial noise

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.55

**Funding** *Christoph Grunau*: Supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 853109). *Václav Rozhoň*: Supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 853109).

**Acknowledgements** We would like to thank Mohsen Ghaffari for many helpful comments.

## 1 Introduction

The $k$-means problem is a classical problem in computer science: given a *point set* $X \subseteq \mathbb{R}^d$ consisting of $n$ points and a parameter $k$, we are asked to return a set of $k$ *clusters* with corresponding cluster centers $C \subseteq \mathbb{R}^d$ so as to minimize the sum of the squared distances of points of $X$ with respect to their closest cluster center in $C$. Formally, we are asked to minimize the function $\varphi(X, C)$ defined by $\varphi(x, C) = \min_{c \in C} ||x - c||^2$ for a single point $x$ and as $\varphi(X, C) = \sum_{x \in X} \varphi(x, C)$ for a set of points.

There exists some fixed constant $c > 1$ such that it is NP-hard to find a $c$-approximate solution to the $k$-means objective [2, 4]. On the other hand, a substantial amount of work has been devoted to finding polynomial time algorithms with a good approximation guarantee, with the currently best approximation ratio being 5.912 [12]. On the practical side, the celebrated clustering algorithm $k$-means++ by Arthur and Vassilvitskii [3] is one of the classical algorithms for the $k$-means problem. Due to its simplicity, it is widely used in practice, for example in the well-known Python Scikit-learn library [18]. It is also very appealing from the theoretical perspective, as it returns a solution that is $O(\log k)$-approximate, in expectation.


The $k$-means++ algorithm (Algorithm 1 with $\varepsilon = 0$) is indeed very simple: we sample $C \subseteq X$ in $k$ steps. The first center is taken as a uniformly random point of $X$. To get each subsequent center, we always first compute the current costs $\varphi(x, C_i)$ for each $x \in X$; then we sample each point of $X$ as the next center with probability proportional to $\varphi(x, C_i)$.

In [10], the authors made an intriguing observation: the classical analysis of the algorithm by Arthur and Vassilvitski [3] fails to work if we allow small errors in the sampling probabilities. That is, consider Algorithm 1: this is the $k$-means++ algorithm, however, with an additional small positive parameter $\varepsilon$. In every step, before we sample, we allow an adversary to perturb the sampling distribution such that the multiplicative change of each probability is within $1 \pm \varepsilon$ of its original value.

**Algorithm 1** $(1+\varepsilon)$-noisy $k$-means++.

Input: $X$, $k$, $0 \le \varepsilon < 1/2$

1: Sample $x \in X$ w.p. in $\left[(1-\varepsilon)\cdot\frac{1}{n}, (1+\varepsilon)\cdot\frac{1}{n}\right]$, set $C_1 = \{x\}$.
2: **for** $i \leftarrow 0, 1, \ldots, k-1$ **do**
3: Sample $x \in X$ w.p. in $\left[(1-\varepsilon)\cdot\frac{\varphi(x,C_i)}{\varphi(X,C_i)}, (1+\varepsilon)\cdot\frac{\varphi(x,C_i)}{\varphi(X,C_i)}\right]$ and set $C_{i+1} = C_i \cup \{x\}$.
**return** $C := C_k$

Does the noisy $k$-means++ algorithm retain the original guarantees? This question is natural since in every implementation, there are small numerical errors associated with the distance computations made by Algorithm 1. It would be shocking if these errors could substantially affect the quality of the algorithm's output! From a more theoretical perspective, the authors of [10] considered this problem as a first step towards understanding other questions related to the $k$-means++ algorithm, in particular the analysis of the *greedy* variant of $k$-means++, a related algorithm later analyzed in [14].

Going back to noisy $k$-means++, the authors of [10] proved that Algorithm 1 remains $O(\log^2 k)$-approximate even for small constant $\varepsilon$ (think e.g. $\varepsilon = 0.01$). In this paper, we improve their analysis to recover the tight $O(\log k)$-approximation guarantee. That is, we show that the adversarial noise worsens the approximation guarantee by at most a constant multiplicative factor.

▶ **Theorem 1.** *Algorithm 1 is $O(\log k)$-approximate, in expectation.*

▶ Remark 2. It would be interesting to see an analysis of the approximation ratio of Algorithm 1 that would be within a $1 + O(\varepsilon)$-factor of the classical $k$-means++ analysis from [3], or a counterexample showing this is not possible. In our analysis, we lose a very large constant factor even for very small $\varepsilon$.

**Related Work.** There is a lot of work related to the $k$-means++ algorithm, both improving the algorithm or its analysis [16, 11, 1, 20, 17, 10, 14] and adapting it to other setups [8, 6, 19, 17, 5, 7, 9, 15].

## 2 Reduction to a Sampling Game

To analyze Algorithm 1, the authors of [10] follow the proof of [3] (more precisely, they follow the proof from [13]) and show that most arguments of that proof, in fact, work even in the adversarial noise scenario. The part of the proof that does not generalize from $\varepsilon = 0$ to $\varepsilon > 0$ can be distilled into a simple sampling process that we analyze in this paper. We next describe this process and state its relation to the analysis of noisy $k$-means++ (cf. the discussion on page 15 of [10]).

▶ **Definition 3** ($(1+\varepsilon)$-adversarial sampling process). *Let $0 < \varepsilon < 1/2$. We define the* $(1+\varepsilon)$-adversarial sampling process *as follows. At the beginning, there is a set $E_0$ of $k$ elements where each element $e \in E_0$ has some nonnegative weight $w_0(e)$. The process has $k$ rounds where in each round, we form the new set $E_{i+1}$ from $E_i$ as follows:*

1. *We define the distribution $D_i$ over $E_i$ where the probability of selecting $e \in E_i$ is defined as $w_i(e)/\sum_{e\in E_i} w_i(e)$. Next, an adversary chooses an arbitrary distribution $D_i^\varepsilon$ over $E_i$ that satisfies for any $e \in E_i$ that*

$$(1-\varepsilon)P_{D_i}(e) \le P_{D_i^\varepsilon}(e) \le (1+\varepsilon)P_{D_i}(e). \tag{1}$$

   *We sample an element $e_{i+1} \in E_i$ according to $D_i^\varepsilon$ and set $E_{i+1} = E_i \setminus \{e_{i+1}\}$.*
2. *Next, an adversary chooses a new weight function $w_{i+1}(e)$ for every element $e \in E_{i+1}$ as an arbitrary function that satisfies*

$$0 \le w_{i+1}(e) \le w_i(e).$$

We will be interested in the expected average weight of an element after some number of steps in this process, that is, we need to understand the value of $\mathrm{E}\left[\frac{\sum_{e\in E_i} w_i(e)}{k-i}\right]$ for $0 \le i < k$. If $\varepsilon = 0$, one can prove that

$$\mathrm{E}\left[\frac{\sum_{e\in E_i} w_i(e)}{k-i}\right] \le \frac{\sum_{e\in E_{i-1}} w_{i-1}(e)}{k-(i-1)} \tag{2}$$

where the randomness is over the sampling in the $i$-th step (we always regard the adversary as fixed in advance). Why is Equation (2) true? The inequality would clearly hold with equality if the distribution $D_i$ were a uniform one and there was no adversary; we in fact give larger sampling probabilities to heavier elements in $D_i$ and, moreover, the adversary can lower the weights arbitrarily after we sample, but both of these operations can make the left-hand side of Equation (2) only smaller.

However, this monotonic behavior is no longer true for $\varepsilon > 0$. The question that needs to be analyzed as a part of the analysis of noisy $k$-means++ is whether the adversarial choices can make the average size of an element drift so that in the end the left-hand side of Equation (2) is substantially larger than $\sum_{e\in E_0} w_0(e)/k$. More precisely, we will need to bound the following quantity that we call the adversarial advantage.

▶ **Definition 4** (Adversarial advantage). *We say that the adversarial advantage is at most some function $f$ if the following conclusion holds: Consider a $(1+\varepsilon)$-adversarial sampling process on $k$ elements for any $0 < \varepsilon < \frac{1}{2}$, any starting set $E_0$, and any adversary. For any $0 \le i < k$, we have*

$$E\left[\frac{\sum_{e\in E_i} w_i(e)}{k-i}\right] \le f(k) \cdot \frac{\sum_{e\in E_0} w_0(e)}{k}. \tag{3}$$

Although we require the inequality Equation (3) to hold for all $i$, note that for all $0 \le i \le (1-\delta)k$ we can choose $f(k) = 1/\delta$ in Equation (3) and it will be satisfied for those values of $i$ simply because $\sum_{e\in E_i} w_i(e) \le \sum_{e\in E_0} w_0(e)$ is true deterministically. Thus, intuitively, $i = k-1$ is the hardest case.

In [10], the authors proved that if we adapt the analysis of $k$-means++ to the noisy $k$-means++, it only picks up the multiplicative factor of $f(k)$. That is, analyzing the $(1+\varepsilon)$-adversarial sampling process is enough to get an upper bound for noisy $k$-means++. The following theorem is proven in [10] (it is proven only for $f(k) = O(\log k)$, but it directly generalizes to any $f(k)$).

▶ **Theorem 5** (Theorem 2 in [10]). *For any $0 < \varepsilon < 1/2$, $(1+\varepsilon)$-noisy k-means++ is $O(f(k) \cdot \log k)$-approximate, in expectation.*

In Lemma 10 of [10], the authors prove that $f(k) = O(\log k)$. The reason for this is that if an element $e \in E_0$ is $\Theta(\log k)$ times larger than the average size of an element of $E_0$, it will be sampled in the first $k/2$ steps of the process with probability $1 - 1/k^{O(1)}$. Thus, the contribution of elements $\Omega(\log k)$ larger than the average to the left-hand side of Equation (3) is negligible even for $i = k - 1$. Hence, $f(k) = O(\log k)$.

▶ **Lemma 6** (Lemma 10 in [10]). *The adversarial advantage is at most $O(\log k)$.*

Our technical contribution is to show that the adversarial advantage is bounded by $O(1)$.

▶ **Lemma 7.** *The adversarial advantage is at most $O(1)$.*

Theorem 1 then follows from Theorem 5 and Lemma 7.

## 3 Analysis of the Sampling Process

This section is devoted to the proof of Lemma 7. We view the adversary as a function fixed at the beginning of the argument. We start by normalizing the starting weights $w_0$ so that the average at the beginning is one, i.e., from now on we assume that $(\sum_{e \in E_0} w_0(e))/k = 1$. For every $E \subseteq E_i$, we define $w_i(E) = \sum_{e \in E} w_i(e)$ and similarly $P_{D_i^\varepsilon}(E) = \sum_{e \in E} P_{D_i^\varepsilon}(e)$. In every step $i$, we consider the partition $E_i = B_i \sqcup M_i \sqcup S_i$ where $e \in E_i$ is in

1. the big set $B_i$ iff $w_i(e) \geq 80$,
2. the medium set $M_i$ iff $2 < w_i(e) < 80$ and
3. the small set $S_i$ iff $w_i(e) \leq 2$.

The main idea of the analysis is to show that $w_i(B_i) = O(|S_i|)$, and thus $\frac{w_i(E_i)}{k-i} = \frac{O(|S_i|)}{|S_i|+|M_i|+|B_i|} = O(1)$, with probability $1 - e^{-\Omega(|S_i|)}$. This turns out (see the proof of Lemma 7) that this is sufficient to show that the adversarial advantage is $O(1)$, i.e., that $\mathrm{E}\left[\frac{w_i(E_i)}{k-i}\right] = O(1)$.

Roughly speaking, we call an iteration with $\ell$ small elements bad, if the total weight of the big elements is greater than $4\ell$, which intuitively means the average drifted way above 1. In general we use the number of the small elements as our main way to refer to the iterations. Then in Lemma 9 we denote with $\ell_{max}$ the number of small elements at the first bad iteration. Using that the previous iterations were good, and $w_{i_{2\ell}}(B_{i_{2\ell}}) \leq 8\ell$ for the bad iterations (Definition 8), we provide an upper bound on the average element size for the following iterations. Even though this bound is depending on the number of the small elements $\ell$, we show in Lemma 10 that an iteration is bad with probability at most $e^{-\frac{\ell}{40}}$, which is enough to show the constant average in expectation.

The following definition is crucial for our analysis.

▶ **Definition 8.** *For every $\ell \in \{1, 2, \ldots, |S_0|\}$, we define $i_\ell$ as the smallest $i$ for which $|S_i| = \ell$. We refer to a given $\ell \in \{1, 2, \ldots, \lfloor |S_0|/2 \rfloor\}$ as bad if both $w_{i_{2\ell}}(B_{i_{2\ell}}) \leq 8\ell$ and $w_{i_\ell}(B_{i_\ell}) > 4\ell$ and otherwise we refer to $\ell$ as good.*

Note that $i_\ell$ is well-defined in the sense that there has to exist at least one $i$ with $|S_i| = \ell$ for every $\ell \in \{1, 2, \ldots, |S_0|\}$. This follows from $|S_{i+1}| \geq |S_i| - 1$ for every $i \in \{1, 2, \ldots, k-1\}$ and $|S_{k-1}| \leq 1$.

▶ **Lemma 9.** *Let $\ell_{max}$ be defined as the largest $\ell \in \{1, 2, \ldots, \lfloor |S_0|/2 \rfloor\}$ such that $\ell$ is bad, if there exists such an $\ell$, and otherwise let $\ell_{max} = 1$. Then, for every $i \in \{0, 1, \ldots, k-1\}$, we have $\frac{w_i(E_i)}{k-i} \leq 90\ell_{max}$.*

**Proof.** We first prove by induction that $w_i(B_i) \leq \max(4|S_i|, 8\ell_{max})$ for every $i \in \{0, 1, \ldots, k-1\}$. As our base case, we consider any $i$ with $|S_i| \geq |S_0|/2$. Using that the average weight is 1 at the beginning, we get $|S_0| \geq k/2$ by Markov's inequality and therefore $w_i(B_i) \leq k \leq 2|S_0| \leq 4|S_i|$. For our induction step, consider some arbitrary $i$ with $|S_i| < |S_0|/2$. Let $\ell := |S_i|$. First, we consider the case that $\ell_{max} \geq \ell$. In particular, this implies $|S_{i-1}| \leq |S_i| + 1 \leq \ell + 1 \leq \ell_{max} + 1$ and therefore we get by induction that

$$w_i(B_i) \leq w_{i-1}(B_{i-1}) \leq \max(4|S_{i-1}|, 8\ell_{max}) \leq \max(4(\ell_{max}+1), 8\ell_{max}) \leq 8\ell_{max}.$$

Thus, it suffices to consider the case that $\ell > \ell_{max}$, which in particular implies that $\ell$ is good. We have $i_{2\ell} < i_\ell \leq i$ (since $\ell \leq |S_0|/2 \leq i$) and therefore we can assume by induction that $w_{i_{2\ell}}(B_{i_{2\ell}}) \leq \max(4(2\ell), 8\ell_{max}) = 8\ell$. As $\ell$ is good, this implies that $w_{i_\ell}(B_{i_\ell}) \leq 4\ell$ and therefore $w_i(B_i) \leq w_{i_\ell}(B_{i_\ell}) \leq 4\ell = 4|S_i|$. This finishes the induction and thus we indeed have $w_i(B_i) \leq \max(4|S_i|, 8\ell_{max})$ for every $i \in \{0, 1, \ldots, k-1\}$. Therefore,

$$\frac{w_i(E_i)}{k-i} \leq \frac{w_i(E_i)}{|S_i|+|M_i|+|B_i|} \leq \frac{w_i(B_i)}{\max(|S_i|,1)} + \frac{80(|S_i|+|M_i|)}{|S_i|+|M_i|} \leq \max(4, 8\ell_{max}) + 80 \leq 90\ell_{max}.$$

◀

▶ **Lemma 10.** *Let $\ell \in \{1, 2, \ldots, \lfloor |S_0|/2 \rfloor\}$. Then, $\ell$ is bad with probability at most $e^{-\frac{\ell}{40}}$.*

For the proof of Lemma 10, we need the following Chernoff-bound variant.

▶ **Lemma 11** (Chernoff bound). *Let $X_1, \ldots, X_\ell$ be independent Bernoulli-distributed random variables, each equal to one with probability $p$. Then,*

$$P\left(\sum_{i=1}^{\ell} X_i < \frac{p\ell}{2}\right) \leq e^{-p\ell/8}.$$

**Proof of Lemma 10.** Throughout the proof, we assume that $w_{i_{2\ell}}(B_{i_{2\ell}}) \leq 8\ell$. In particular,

$$|B_{i_{2\ell}}| \leq \frac{w_{i_{2\ell}}(B_{i_{2\ell}})}{80} \leq \frac{\ell}{10}.$$

Below, we will define for every $j \in \{1, 2, \ldots, \ell\}$ an indicator variable $X_j$ in such a way that

1. $\mathrm{E}[X_j | X_1, X_2, \ldots, X_{j-1}] \geq \frac{1}{5}$ for every $j \in \{1, 2, \ldots, \ell\}$ and
2. if $X := \sum_{j=1}^{\ell} X_j \geq \frac{\ell}{10}$, then $w_{i_\ell}(B_{i_\ell}) \leq 4\ell$.

The first property implies that $X$ stochastically dominates a random variable $X'$ which is the sum of $\ell$ independent Bernoulli-distributed random variables, each equal to one with probability 1/5. Thus, using Lemma 11, we get

$$\mathrm{P}\left[X < \frac{\ell}{10}\right] \leq \mathrm{P}\left[X' < \frac{\ell}{10}\right] \leq e^{-\frac{\ell}{40}}.$$

Thus, we can now use the second property to deduce that $\ell$ is bad with probability at most $e^{-\frac{\ell}{40}}$. It thus remains to define the random variables and show that they indeed satisfy the two properties. To that end, fix some $j \in \{1, 2, \ldots, \ell\}$. We define $i'_j$ as the smallest $i \in \{i_{2\ell}, i_{2\ell}+1, \ldots, i_\ell - 1\}$ with $|S_i| = 2\ell - j + 1$ and $e_{i+1} \notin M_i$. Note that there exists at least one such $i$ as there exists some $i$ with $|S_i| = 2\ell - j + 1$ and $|S_{i+1}| = 2\ell - j$, and for this $i$ it holds that $e_{i+1} \in S_i$ and therefore $e_{i+1} \notin M_i$. Note that it furthermore holds that $i'_1 < i'_2 < \ldots < i'_\ell$. We set $X_j = 1$ if $w_{i'_j}(B_{i'_j}) \leq 4\ell$ or $e_{i'_j+1} \in B_{i'_j}$ and otherwise we set $X_j = 0$. We start by showing that the second property holds by proving the contrapositive.

To that end, assume that $w_{i_\ell}(B_{i_\ell}) > 4\ell$. In particular, we have for every $j$ that $w_{i'_j}(B_{i'_j}) > 4\ell$. Thus, if $X_j = 1$, we get $e_{i'_j+1} \in B_{i'_j}$ and therefore $|B_{i'_j+1}| \leq |B_{i'_j}| - 1$. As $|B_{i_{2\ell}}| < \frac{\ell}{10}$, we therefore get that $X < \frac{\ell}{10}$, as needed.

It remains to show the first property. To that end, consider any $i$ and assume we have already sampled $e_1, \ldots, e_i$ in an arbitrary manner such that $|S_i| \leq 2\ell$ and $w_i(B_i) \geq 4\ell$. Then, conditioned on $e_{i+1} \notin M_i$, we get $e_{i+1} \in B_i$ with probability at least

$$\frac{D_i^\varepsilon(B_i)}{D_i^\varepsilon(B_i) + D_i^\varepsilon(S_i)} \geq \frac{(1-\varepsilon)w_i(B_i)}{(1-\varepsilon)w_i(B_i) + (1+\varepsilon)w_i(S_i)} \geq \frac{0.5 \cdot 4\ell}{0.5 \cdot 4\ell + 1.5 \cdot 2 \cdot 2\ell} \geq \frac{1}{5}.$$

In particular, this directly implies $\mathrm{E}[X_j | X_1, X_2, \ldots, X_{j-1}] \geq \frac{1}{5}$ for every $j \in \{1, 2, \ldots, \ell\}$. ◀

Finally, we are ready to prove Lemma 7 by combining Lemmas 9 and 10.

**Proof of Lemma 7.** Fix some $i \in \{0, 1, \ldots, k-1\}$. Let $\ell_{max}$ be defined as in Lemma 9. Lemma 9 gives that for every $\ell$ with $Pr[\ell_{max} = \ell] > 0$, we have

$$\mathrm{E}\left[\frac{\sum_{e \in E_i} w_i(e)}{k-i} | \ell_{max} = \ell\right] \leq 90\ell.$$

Moreover, for $\ell > 1$ , we can use Lemma 10 to deduce that $\mathrm{P}[\ell_{max} = \ell] \leq \mathrm{P}[\ell \text{ is bad}] \leq e^{-\frac{\ell}{40}}$. Therefore,

$$\mathrm{E}\left[\frac{\sum_{e \in E_i} w_i(e)}{k-i}\right] \leq \sum_{\ell=1}^{\infty} 90\ell \cdot e^{-\frac{\ell-1}{40}} = O(1).$$ ◀

## References


1. Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 15–28. Springer, 2009.
2. Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248, 2009.
3. David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.
4. Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. The hardness of approximation of euclidean k-means. *arXiv preprint*, 2015. `arXiv:1502.03316`.
5. Olivier Bachem, Mario Lucic, Hamed Hassani, and Andreas Krause. Fast and provably good seedings for k-means. In *Advances in neural information processing systems*, pages 55–63, 2016.
6. Olivier Bachem, Mario Lucic, S Hamed Hassani, and Andreas Krause. Approximate k-means++ in sublinear time. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
7. Olivier Bachem, Mario Lucic, and Andreas Krause. Distributed and provably good seedings for k-means in constant rounds. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 292–300. JMLR. org, 2017.
8. Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.
9. Aditya Bhaskara, Sharvaree Vadgama, and Hong Xu. Greedy sampling for approximate clustering in the presence of outliers. *Advances in Neural Information Processing Systems*, 32, 2019.

**10** Anup Bhattacharya, Jan Eube, Heiko Röglin, and Melanie Schmidt. Noisy, greedy and not so greedy k-means++. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

**11** Davin Choo, Christoph Grunau, Julian Portmann, and Václav Rozhon. k-means++: few more steps yield constant approximation. In *International Conference on Machine Learning*, pages 1909–1917. PMLR, 2020.

**12** Vincent Cohen-Addad, Hossein Esfandiari, Vahab Mirrokni, and Shyam Narayanan. Improved approximations for euclidean $k$-means and $k$-median, via nested quasi-independent sets, 2022. doi:10.48550/ARXIV.2204.04828.

**13** Sanjoy Dasgupta. Lecture 3 – algorithms for k-means clustering, 2013, accessed May 8th, 2019.

**14** Christoph Grunau, Ahmet Alper Özüdoğru, Václav Rozhoň, and Jakub Tětek. A nearly tight analysis of greedy k-means++. *arXiv preprint*, 2022. arXiv:2207.07949.

**15** Christoph Grunau and Václav Rozhoň. Adapting $k$-means algorithms for outliers, 2020. doi:10.48550/arXiv.2007.01118.

**16** Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In *International Conference on Machine Learning*, pages 3662–3671, 2019.

**17** Konstantin Makarychev, Aravind Reddy, and Liren Shan. Improved guarantees for k-means++ and k-means++ parallel. *Advances in Neural Information Processing Systems*, 33:16142–16152, 2020.

**18** F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

**19** Václav Rozhoň. Simple and sharp analysis of k-means||. In *International Conference on Machine Learning*, pages 8266–8275. PMLR, 2020.

**20** Dennis Wei. A constant-factor bi-criteria approximation guarantee for k-means++. In *Advances in Neural Information Processing Systems*, pages 604–612, 2016.

# Convergence to Lexicographically Optimal Base in a (Contra)Polymatroid and Applications to Densest Subgraph and Tree Packing

**Elfarouk Harb**
University of Illinois at Urbana Champaign, IL, USA

**Kent Quanrud**
Purdue University, West Lafayette, IN, USA

**Chandra Chekuri**
University of Illinois at Urbana Champaign, IL, USA

## Abstract

Boob et al. [7] described an iterative peeling algorithm called Greedy++ for the Densest Subgraph Problem (DSG) and conjectured that it converges to an optimum solution. Chekuri, Qaunrud and Torres [10] extended the algorithm to supermodular density problems (of which DSG is a special case) and proved that the resulting algorithm Super-Greedy++ (and hence also Greedy++) converges. In this paper we revisit the convergence proof and provide a different perspective. This is done via a connection to Fujishige's quadratic program for finding a lexicographically optimal base in a (contra) polymatroid [18], and a noisy version of the Frank-Wolfe method from convex optimization [17, 25]. This yields a simpler convergence proof, and also shows a stronger property that Super-Greedy++ converges to the optimal dense decomposition vector, answering a question raised in Harb et al. [24]. A second contribution of the paper is to understand Thorup's work on ideal tree packing and greedy tree packing [46, 47] via the Frank-Wolfe algorithm applied to find a lexicographically optimum base in the graphic matroid. This yields a simpler and transparent proof. The two results appear disparate but are unified via Fujishige's result and convex optimization.

**2012 ACM Subject Classification** Networks → Network algorithms; Mathematics of computing → Graph algorithms

**Keywords and phrases** Polymatroid, lexicographically optimum base, densest subgraph, tree packing

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.56

**Related Version** *Full Version*: https://arxiv.org/abs/2305.02987

**Funding** *Elfarouk Harb*: Supported in part by NSF grants CCF-2028861 and CCF-1910149.
*Kent Quanrud*: Supported in part by NSF grant CCF-2129816.
*Chandra Chekuri*: Supported in part by NSF grants CCF-2028861 and CCF-1910149.

## 1 Introduction

In this paper we consider iterative greedy algorithms for two different combinatorial optimization problems and show that the convergence of these algorithms can be understood by combining two general tools, one coming from the theory of submodular functions, and the other coming from convex optimization. This yields simpler proofs via a unified perspective, while also yielding additional properties that were previously unknown.

**Densest subgraph and supermodularity.** We start with the problem that motivated this work, namely, the densest subgraph problem (DSG). The input to DSG is an undirected graph $G = (V, E)$ with $m = |E|$ and $n = |V|$. The goal is to return a subset $S \subseteq V$ that maximizes $\frac{|E(S)|}{|S|}$ where $E(S) = \{uv \in E : u, v \in S\}$ is the set of edges with both end points


in $S$. Throughout the paper, we let $\lambda(G) = \frac{|E(G)|}{|V(G)|}$ denote the density of graph $G(V, E)$. We treat the unweighted case for simplicity; all the results generalize to edge-weighted graphs. Goldberg [22] and Picard and Queyranne [37] showed that DSG can be efficiently solved via a reduction to the $s$-$t$ maximum-flow problem.

A different connection that shows polynomial-time solvability of DSG is important to this paper. Consider a real-valued set function $f : 2^V \to \mathbb{R}_+$ defined over the vertex set $V$, where $f(S) = |E(S)|$. This function is *supermodular*. A function $f$ is supermodular iff $-f$ is *submodular*. A real-valued set function $f : 2^V \to \mathbb{R}$ is submodular iff $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq B$. Submodular and supermodular set functions are fundamental in combinatorial optimization – see [41, 19].

Coming back to DSG, maximizing $|E(S)|/|S|$ is equivalent to finding the largest $\lambda$ such that $\lambda|S| - |E(S)| \geq 0$ for all $S \subseteq V$. This corresponds to minimizing the submodular set function $g$ where $g(S) = \lambda|S| - |E(S)|$. A classical result in combinatorial optimization is that the minimum of a submodular set function can be found in polynomial-time in the value oracle setting [41]. Thus, DSG can be solved via reduction to submodular set function minimization and binary search. The preceding connection also motivates the definition of a generalization of DSG called the densest supermodular set problem (DSS) [10]. The input is a non-negative supermodular function $f : 2^V \to \Re_+$, and the goal is to find $S \subseteq V$ that maximizes $\frac{f(S)}{|S|}$. DSS is polynomial-time solvable via submodular set function minimization. DSG, DSS and its variants have several applications in practice, and they are routinely used in graph and network analysis to find dense clusters or communities. We refer the reader to the extensive literature on this topic [32, 7, 14, 48, 43, 1, 49, 16, 34, 39, 6, 27, 2, 42, 30, 28]. DSG is also of interest in algorithms via its connection to arboricity and related notions – see [40, 13] for recent work.

**Faster algorithms, Greedy and Greedy++.** Although DSG is polynomial-time solvable via maxflow or submodular function minimization, the corresponding algorithms are not yet practical for the large graphs that arise in many applications; this is despite the fact that we now have very fast theoretical algorithms for maxflow and mincost flow [12]. For this reason there has been considerable interest in fast (approximation) algorithms. More than 20 years ago Charikar [9] showed that a simple "peeling" algorithm (Greedy) yields a 1/2-approximation for DSG. An ordering of the vertices as $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ is computed as follows: $v_{i_1}$ is a vertex of minimum degree in $G$ (ties broken arbitrarily), $v_{i_2}$ is a minimum degree vertex in $G - v_{i_1}$ and so on[1]. After creating the ordering, the algorithm picks the best suffix, in terms of density, among the $n$-possible suffixes of the ordering. Charikar also developed a simple exact LP relaxation for DSG. Charikar's results have been quite influential. Greedy can be implemented in (near)-linear time and has also been adapted to other variants. The LP relaxation has also been used in several algorithms that yield a $(1 - \epsilon)$-approximate solution [5, 8], and has led to a flow-based $(1 - \epsilon)$-approximation [10]. More recently, Boob et al. [7] developed an algorithm called Greedy++ that is based on combining Greedy with ideas from multiplicative weight updates (MWU); the algorithm repeatedly applies a simple peeling algorithm with the first iteration coinciding with Greedy but later iterations depending on a weight vector that is maintained on the vertices – the formal algorithm is described in a later section. The advantage of the algorithm is its simplicity, and Boob et al. [7] showed that it has good empirical performance. Moreover

[1] This peeling order is the same as the one used to create the so-called core decomposition of a graph [33] and the Greedy algorithm itself was suggested by Asahiro et al. [4].

they conjectured that GREEDY++ converges to a $(1-\epsilon)$-approximation in $O(1/\epsilon^2)$ iterations. Although their strong conjecture is yet unverified, Chekuri, Quanrud and Torres [10] proved that GREEDY++ converges in $O(\frac{\Delta \log |V|}{\epsilon^2 \lambda^*(G)})$ iterations where $\Delta$ is the maximum degree of $G$ and $\lambda^*(G)$ is the optimum density.

The convergence proof in [10] is non-trivial and relies crucially in considering DSS and supermodularity. [10] shows that GREEDY and GREEDY++ can be generalized to SUPERGREEDY and SUPERGREEDY++ for DSS, and that SUPERGREEDY++ converges to a $(1-\epsilon)$-approximation solution in $O(\alpha_f/\epsilon^2)$ iterations where $\alpha_f$ depends (only) on the function $f$.

**Dense subgraph decomposition and connections.** As we discussed, DSG is a special case of DSS and hence DSG inherits certain nice structural properties from supermodularity. One of these is the fact that the vertex set $V$ of every graph $G = (V, E)$ admits a unique decomposition into $S_1, S_2, \ldots, S_k$ for some $k$ using the following procedure: $S_1$ is the vertex set of the *unique maximal* densest subgraph, $S_2$ is the unique maximal densest subgraph after "contracting" $S_1$, and so on. The existence of such a unique decomposition is more transparent in the setting of DSS. The fact that there is a *unique* maximal densest set $S_1$ follows from supermodularity; if $A$ and $B$ are optimum dense sets then so is $A \cup B$. One can then consider a new supermodular function $f_{S_1} : 2^{V-S_1} \to \mathbb{R}$ defined over $V - S_1$ where $f_{S_1}(A) = f(S_1 \cup A) - f(S_1)$ for all $A \subseteq V - S_1$. The new function is also supermodular. Then $S_2$ is the unique maximal densest set for $f_{S_1}$. We iterate this process until we obtain an empty set. The decomposition also allows us to assign a density value $\lambda_v$ to each $v \in V$ (which corresponds to the density of the set when $v$ is in the maximal set). We call this the density vector associated with $f$. Dense decompositions follow from the theory of principal partitions of submodular functions [35, 36, 20]. In the context of graphs and DSG this was rediscovered by Tatti and Gionis who called it the locally-dense decomposition [45, 44], and gave algorithms for computing it. Subsequently, Danisch et al. [14] applied the well-known Frank-Wolfe algorithm for constrained convex optimization to a quadratic program derived from Charikar's LP relaxation for DSG. More recently, Harb et al. [24] obtained faster algorithms for computing the dense decomposition in graphs via Charikar's LP; they used a different method called FISTA for constrained convex optimization based on acceleration. Although DSS was not the main focus, [24] also made an important connection to Fujishige's result on lexicographically optimal base in polymatroids [18] which elucidated the work of Danisch et al. on DSG. We describe this next.

**Lexicographical optimal base and dense decomposition.** We briefly describe Fujishige's result [18] and its connection to dense decompositions. Let $f : 2^V \to \mathbb{R}_+$ be a monotone submodular set function ($f(A) \leq f(B)$ if $A \subset B$) that is also normalized ($f(\emptyset) = 0$). Following Edmonds, the polymatroid associated with $f$, denote by $P_f$ is the polyhedron $\{x \in \mathbb{R}^V \mid x \geq 0, x(S) \leq f(S) \quad \forall S \subseteq V\}$, where $x(S) = \sum_{i \in S} x_i$. The base polyhedron associated with $f$, denote by $B_f$, is the polyhedron $P_f \cap \{x \in \mathbb{R}^V \mid x(V) = f(V)\}$ obtained by intersecting $P_f$ with the equality constraint $x(V) = f(V)$. Each vector $x$ in $B_f$ is called a base. If $f$ is a monotone normalized *supermodular* function, we consider the contrapolymatroid $P_f = \{x \in \mathbb{R}^V \mid x \geq 0, x(S) \geq f(S) \quad \forall S \subseteq V\}$ (the inequalities are reversed), and similarly $B_f$ is the base contrapolymatroid obtained by intersecting $P_f$ with equality constraint $x(V) = f(V)$. Fujishige proved that there exists a unique lexicographically minimal base in any polymatroid, and morover it can found by solving the quadratic program: $\min \sum_v x_v^2$ s.t $x \in B_f$. In the context of supermodular functions, one obtains a similar result;

the quadratic program $\min \sum_v x_v^2$ s.t $x \in B_f$ where $B_f$ is contrapolymatroid associated with $f$ has a unique solution. As observed explicity in [24], the lexicographically optimal base gives the dense decomposition vector for DSS. That is, if $x^*$ is the optimal solution to the quadratic program then for each $v$, $x_v^* = \lambda_v$. In particular, as noted in [24], one can apply the well-known Frank-Wolfe algorithm to the quadratic program and it converges to the dense decomposition vector. As we will see later, each iteration corresponds to finding a maximum weight base in a contrapolymatroid which is easy to find via the greedy algorithm.

**(Ideal) Tree packings in graphs and the Tutte–Nash-Williams theorem.** Our discussion so far focused on DSG. Now we describe a different problem on graphs and relevant background. Our goal is to present a unified perspective on these two problems. The well-known Tutte–Nash-Williams theorem in graph theory (see [41]) establishes a min-max result for the maximum number of edge-disjoint spanning trees in a multi-graph $G$. Given an undirected graph $G = (V, E)$, and a partition $P$ of the vertices, let $E(P)$ denote the set of edges crossing the partition. The strength of a partition $P$ is defined as $\frac{|E(P)|}{|P|-1}$. Let $\mathcal{T}(G)$ denote all possible spanning trees of $G$. Let $\tau^*(G)$ denote the maximum number of edge-disjoint spanning trees in $G$. Then $\tau^*(G) = \min_P \lfloor \frac{|E(P)|}{|P|-1} \rfloor$. Further, if we define $\tau(G)$ to be the maximum *fractional* packing of spanning trees, then the floor can be removed and we have $\tau(G) = \min_P \frac{|E(P)|}{|P|-1}$. We note that the graph theoretic result is a special case of matroid base packing. Tree packings are useful for a number of applications. In particular, Karger [26] used tree packings and other ideas in his well-known near-linear randomized algorithm for computing the global minimum cut of a graph. We are mainly concerned here with Thorup's work in [46, 47] that was motivated by dynamic mincut and $k$-cut problems. He defined the so-called *ideal* edge loads and ideal tree packing (details in later section) by recursively decomposing the graph via Tutte–Nash-Williams partitions [46]. He also proved that a simple iterative greedy tree packing algorithm converges to the ideal loads [47]. He used the approximate ideal tree packing to obtain new deterministic algorithms for the $k$-cut problem, and his approach has been quite influential in a number of subsequent results [21, 11, 31, 29, 23]. Thorup obtained his tree packing result from first principles. We ask: is there a connection between ideal tree packing and DSG?

## 1.1 Contributions of the paper

This paper has two main contributions. The first is a new proof of the convergence of SuperGreedy++ for DSS. Our proof is based on showing that SuperGreedy++ can be viewed as a "noisy" or "approximate" variant of the Frank-Wolfe algorithm applied to the quadratic program defined by Fujishige. The advantage of the new proof is twofold. First, it shows that SuperGreedy++ not only converges to a $(1-\epsilon)$-approximation to the densest set, but that in fact it converges to the densest decomposition vector. This was empirically observed in [24] for DSG, and was left as an open problem to resolve. The proof in [10] on convergence of SuperGreedy++ is based on the MWU method via LPs, and does not exploit Fujishige's result which is key to the stronger property that we prove here. Second, the proof connects two powerful tools directly and at a high-level: Fujishige's result on submodular functions, and a standard method for constrained convex optimization.

▶ **Theorem 1.** *Let $b^*$ be the dense decomposition vector for a non-negative monotone supermodular set function $f : 2^V \to \mathbb{R}_+$ where $|V| = n$. Then, SuperGreedy++ converges in $O(\alpha_f/\epsilon^2)$ iterations to a vector $b$ such that $||b - b^*||_2 \leq \epsilon$, where $\alpha_f$ depends only on $f$. For a graph with $m$ edges and $n$ vertices, Greedy++ converges in $O(mn^2/\epsilon^2)$ iterations for unweighted multigraphs.*

▶ Remark 2. The new convergence gives a weaker bound than the one in [10] in terms of convergence to a $(1-\epsilon)$ *relative* approximation to the maximum density. However, it gives a strong *additive* guarantee to the *entire* dense decomposition vector.

Our second contribution builds on our insights on DSG and DSS, and applies it towards understanding ideal tree packing and greed tree packing. We connect the ideal tree packing of Thorup to the dense decomposition associated with the rank function of the underlying graphic matroid (which is submodular). We then show that greedy tree packing algorithm can be viewed as the Frank-Wolfe algorithm applied to the quadratic program defined by Fujishige, and this easily yields a convergence guarantee.

▶ **Theorem 3.** *Let $G = (V, E)$ be a graph. The ideal edge load vector $\ell^* : E \to \mathbb{R}_+$ for $G$ is given by the lexicographically minimal base in the polymatroid associated with the rank function of the graphic matroid of $G$. The Frank-Wolfe algorithm with step size $\frac{1}{k+1}$, when applied to the quadratic program for computing the lexicographically minimal base in the graphic matroid of $G$, coincides with the greedy tree packing algorithm. For unweighted graphs on $m$ edges, the generic analysis of Frank-Wolfe method's convergence shows that greedy tree packing converges to a load vector $\ell : E \to \mathbb{R}_+$ such that $||\ell - \ell^*||_2 \leq \epsilon$ in $O(\frac{m \log(m/\epsilon)}{\epsilon^2})$ iterations. The standard step size algorithm converges in $O(\frac{m}{\epsilon^2})$ iterations.*

▶ Remark 4. Although the algorithm is the same (greedy tree packing), Thorup's analysis guarantees a strongly polynomial-bound even in the capacitated case [47]. However we obtain a stronger additive guarantee via a *generic* Frank-Wolfe analysis and our analysis has a $1/\epsilon^2$ dependence while Thorup's has a $1/\epsilon^3$ dependence. We give a more detailed comparison in Section 5.

**Organization.** The rest of the paper is devoted to proving the two theorems. The paper relies on tools from theory of submodular functions and an adaptation of the analysis of Frank-Wolfe. We first describe the relevant background and then prove the two results in separate sections. Due to space constraints, most of the proofs are provided in the full version.

## 2 Background on Frank-Wolfe algorithm and a variation

Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function. Consider the problem of $\min_{x \in \mathcal{D}} f(x)$. Frank-Wolfe [17] is a first order method and it relies on access to a linear minimization oracle, LMO, for $f$ that can answer $\text{LMO}(w) = \arg\min_{s \in \mathcal{D}} \langle s, \nabla f(w) \rangle$ for any given $w \in \mathcal{D}$. In several applications such oracles with fast running times exist. Given $f, \mathcal{D}$ as above, the Frank-Wolfe algorithm is an iterative algorithm that converges to the minimizer $\mathbf{x}^* \in \mathcal{D}$ of $f$. See Algorithm 1. The algorithm starts with a guess of the minimizer $b^{(0)} \in \mathcal{D}$. In each iteration, it finds a direction $d^{(k+1)}$ to move towards by calling the linear minimization oracle on the current guess $b^{(k)}$. It then moves slightly towards that direction using a convex combination to ensure that the new point is in $\mathcal{D}$. The amount the algorithm moves towards the new direction decreases as $k$ increases signifying the "confidence" in its current guess as the minimizer.

The original convergence analysis for the Frank-Wolfe algorithm is from [17]. Jaggi [25] gave an elegant and simpler analysis. His analysis characterizes the convergence rate in terms of the *curvature constant* $C_f$ of the function $f$.

**Algorithm 1** FRANK-WOLFE-ORIGINAL.

Initialize $b^{(0)} \in \mathcal{D}$
**for** $k \leftarrow 0$ to $T-1$ **do**
  $\gamma \leftarrow \frac{2}{k+2}$
  $d^{(k+1)} \leftarrow \arg\min_{s \in \mathcal{D}}(\langle s, \nabla f(b^{(k)})\rangle)$ ▷ Call oracle on $b^{(k)}$
  $b^{(k+1)} \leftarrow (1-\gamma)b^{(k)} + \gamma d^{(k+1)}$
**return** $b^{(T)}$

▶ **Definition 5.** *Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function. The curvature constant $C_f$ of $f$ is defined as*

$$C_f = \sup_{x,s \in D, \gamma \in [0,1], y = x+\gamma(s-x)} \frac{2}{\gamma^2}(f(y) - f(x) - \langle y - x, \nabla f(x)\rangle).$$

▶ **Definition 6.** *Let $g : \mathcal{D} \to \Re$ be a differentiable function. Then $g$ is Lipschitz with constant $L$ if for all $x, y \in \mathcal{D}$, $\|g(\mathbf{x}) - g(\mathbf{y})\|_2 \leq L \|x - y\|_2$.*

Let $\text{diam}(\mathcal{D}) = \max_{x,y \in \mathcal{D}} \|x - y\|_2$ be the diameter of $\mathcal{D}$. One can show that $C_f \leq L \cdot \text{diam}(\mathcal{D})^2$ where $L$ is the Lipschitz constant of $\nabla f$.

▶ **Theorem 7** ([25])**.** *Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function with minimizer $\mathbf{b}^*$. Let $\mathbf{b}^{(k)}$ denote the guess on the $k$-th iteration of the Frank-Wolfe algorithm. Then $f(\mathbf{b}^{(k)}) - f(\mathbf{b}^*) \leq \frac{2C_f}{k+2}$.*

Jaggi's proof technique can be used to prove the convergence rate of "noisy/approximate" variants of the Frank-Wolfe algorithm. This motivates the following definition. An $\epsilon$-*approximate linear minimization oracle* is an oracle that for any $\mathbf{w} \in \mathcal{D}$, returns $\hat{\mathbf{s}}$ such that $\langle \hat{\mathbf{s}}, \nabla f(\mathbf{w})\rangle \leq \langle \mathbf{s}^*, \nabla f(\mathbf{w})\rangle + \epsilon$, where $s^* = \text{LMO}(\mathbf{w})$. While an efficient *exact* linear minimization oracle exists in some applications, in others one can only $\epsilon$-approximate it (using numerical methods or otherwise). Jaggi's proof technique extends to show that an approximate linear minimization oracles suffices for convergence as long as the approximation quality improves with the iterations. Suppose the oracle, in iteration $k$, provides a $\frac{\delta C_f}{k+2}$-approximate solution where $\delta > 0$ is some fixed constant. The convergence rate will only deteriorate by a $(1+\delta)$ multiplicative factor. Qualitatively, this says that we can afford to be inaccurate in computing the Frank-Wolfe direction in early iterations, but the approximation should approach $\text{LMO}(b^{(k)})$ as $k \to \infty$.

Another question of interest is the resilience of the Frank-Wolfe algorithm to changes in the learning rate $\gamma_k = \frac{2}{k+2}$. Indeed, the variants we will look at will *require* $\gamma_k = \frac{1}{k+1}$. Jaggi's proof can again be adapted to handle this case, with only an $O(\log k)$ multiplicative deterioration in the convergence rate. We state the following theorem whose proof we defer to the appendix.

▶ **Theorem 8.** *Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function with minimizer $\mathbf{b}^*$. Suppose instead of computing $\mathbf{d}^{(k+1)}$ by calling LMO$(\mathbf{b}^{(\mathbf{k})})$ in iteration $k$, we call a $\frac{\delta C_f}{k+2}$-approximate linear minimization oracle, for some fixed $\delta > 0$. Also, suppose instead of using $\gamma_k = \frac{2}{k+2}$, we use $\gamma_k = \frac{1}{k+1}$ as a step size. Then $f(\mathbf{b}^{(k)}) - f(\mathbf{b}^*) \leq \frac{2C_f(1+\delta)H_{k+1}}{k+1}$, where $H_n$ is the $n$-th Harmonic term.*

We refer to the variant of Frank-Wolfe algorithm, as described by Theorem 8, as *noisy* Frank-Wolfe.

## 3 Sub and supermodular functions, and dense decompositions

We already defined submodular and supermodular set functions, polymatroids and contrapolymatroids. We restrict attention to functions satisfying $f(\emptyset) = 0$ which together with supermodularity and non-negativity implies monotonocity, that is, $f(A) \leq f(B)$ for $A \subseteq B$. An alternative definition of submodularity is via diminishing marginal values. We let $f(v \mid A) = f(A \cup \{v\}) - f(A)$ denote the marginal value of $v$ to $A$. Submodularity is equivalent to $f(v \mid A) \geq f(v \mid B)$ whenever $A \subseteq B$ and $v \in V \setminus B$; the inequality is reversed for supermodular set functions. We need the following simple lemma.

▶ **Lemma 9.** *For a submodular function $f : 2^V \to \Re$, the function $g(X) = f(V) - f(V \setminus X)$ is supermodular. In particular if $f$ is a normalized monotone submodular function then $g$ is a normalized monotone supermodular function.*

**Deletion and contraction, and non-negative summation.** Sub and supermodular functions are closed under a few simple operations. Given $f : 2^V \to \mathbb{R}$, restricting it to a subset $V'$ corresponds to deleting $V \setminus V'$. Given $A \subset V$, contracting $f$ to $A$ yields the function $g : 2^{V \setminus A} \to \mathbb{R}$ where $g(X) = g(X \cup A) - g(A)$. Given two functions $f$ and $g$ we can take their non-negative sum $af + bg$ where $a, b \geq 0$. Monotonicity and normalization is also preserved under these operations.

### 3.1 Dense decompositions for submodular and supermodular functions

Following the discussion in the introduction, we are interested in decompositions of supermodular and submodular functions. Dense decompositions follow from the theory of principal partitions of submodular functions that have been explored extensively. We refer the reader to Fujishige's survey [20] as well as Naraynan's work [35, 36]. The standard perspective comes from considering the minimizers of the function $f_\lambda$ for a scalar $\lambda$ where $f_\lambda(S) = f(S) - \lambda|S|$. As $\lambda$ varies from $-\infty$ to $\infty$ the minimizers change only at a finite number of break points. In this paper we are interested in the notion of density, in the form of ratios, for non-negative submodular and supermodular functions. For this reason we follow the notation from recent work [44, 14, 10, 24] and state lemmas in a convenient form, and provide proofs in the appendix for the sake of completeness.

**Supermodular function dense decomposition.** The basic observation is the following.

▶ **Lemma 10.** *Let $f : 2^V \to \Re_+$ be a non-negative supermodular set function. There exists a* unique maximal *set $S \subseteq V$ that maximizes $\frac{f(S)}{|S|}$.*

The preceding lemma can be used in a simple fashion to derive the following corollary (this was explicitly noted in [10] for instance).

▶ **Corollary 11.** *Let $f : 2^V \to \Re_+$ be a non-negative supermodular set function. There is a unique partition $S_1, S_2, \ldots, S_h$ of $V$ with the following property. Let $V_i = V - \cup_{j<i} S_j$ and let $A_i = \cup_{j<i} S_i$. Then, for each $i = 1$ to $h$, $S_i$ is the unique maximal densest set for the function $f_{D_i} : 2^{V_i} \to \mathbb{R}_+$. Moroever, letting $\lambda_i$ be the optimum density of $f_{D_i}$, we have $\lambda_1 > \lambda_2 \ldots > \lambda_h$.*

Based on the preceding corollary, we can associated with each $v \in V$ a value $\lambda(v)$: $\lambda(v) = \lambda_i$ where $v \in S_i$. See Figure 1 (full version) for an example of a dense decomposition of the function $f(S) = |E(S)|$.

**Dense decomposition for submodular functions.** We now discuss submodular functions. We consider two variants. We start with a basic observation.

▶ **Lemma 12.** *Let $f : 2^V \to \Re_+$ be a monotone non-negative submodular set function such that $f(v) > 0$ for all $v \in V$. There is a unique minimal set $S \subseteq V$ that minimizes $\frac{|V|-|S|}{f(V)-f(S)}$ for submodular function $f$.*

Consider the following variant of a decomposition of $f$. We let $S_0 = V$ and find $S_1$ as the unique *minimal* set $S \subseteq V$ that minimizes $\frac{|V|-|S|}{f(V)-f(S)}$. Then we "delete" $\hat{S}_1 = V \setminus S_1$, and find the minimal set $S_2 \subseteq S_1$ that minimizes $\frac{|S_1|-|S|}{f(S_1)-f(S)}$. In iteration $i$, we find the unique minimal set $S_i \subset S_{i-1}$ that minimizes $\frac{|S_{i-1}|-|S_i|}{f(S_{i-1})-f(S_i)}$. Notice that $S_k \subset S_{k-1} \subset ... \subset S_1 \subset V$. We say the relative density of $\hat{S}_i = S_{i-1} \setminus S_i$ is $\lambda_i = \frac{|S_{i-1}|-|S_i|}{f(S_{i-1})-f(S_i)}$. For $u \in \hat{S}_i$, we say the density of $u$ is $\lambda_u = \lambda_i$. Hence the dense decomposition of $f$ is $\hat{S}_1, ..., \hat{S}_k$ with densities $\lambda_1, \ldots, \lambda_k$. We refer to this decomposition as the first variant which is based on deletions.

We now describe a second dense decomposition for submodular functions. Let $f : 2^V \to \mathbb{R}_+$ be a monotone submodular function. Consider the supermodular function $g : 2^V \to \mathbb{R}_+$ where $g(X) = f(V) - f(V \setminus X)$ for all $X \subseteq V$. From Lemma 9, $g$ is monotone supermodular. We can then apply Corollary 11 to obtain a dense decomposition of $g$. Let $T_1, T_2, \ldots, T_{k'}$ be the unique decomposition obtained by considering $g$ and let $\hat{\lambda}_1, ..., \hat{\lambda}_{k'}$ be the corresponding densities. Note that this second decomposition is based on contractions.

Not too surprisingly, the two decompositions coincide, as we show in the next theorem. The main reason to consider them separately is for technical ease in applications where one or the other view is more natural.

▶ **Theorem 13.** *Let $\hat{S}_1, ..., \hat{S}_k$ be a dense decomposition (using deletion variant) of a submodular function $f$ with densities $\lambda_i, \ldots, \lambda_k$. Let $T_1, ..., T_{k'}$ be a dense decomposition (using contraction variant) of the same function with densities $\hat{\lambda}_1, ..., \hat{\lambda}_{k'}$. We have (i) $k' = k$, (ii) $\hat{S}_1, ...\hat{S}_k$ is exactly $T_1, ..., T_k$, and (iii) $\hat{\lambda}_i = \frac{1}{\lambda_i}$ for $1 \leq i \leq k$.*

## 3.2 Fujishige's results on lexicographically optimal bases

Fujishige [18] gave a polyhedral view of the dense decomposition which is the central ingredient in our work. He stated his theorem for polymatroids, however, it can be easily generalized to contrapolymatroids. We restrict attention to the unweighted case for notational ease – [18] treats the weighted case.

Vectors in $\mathbb{R}^n$ can be totally ordered by sorting the coordinates in increasing order of value and considering the lexicographical ordering of the two sorted sequences of length $n$. In the following, for $a, b \in \mathbb{R}^n$ we use $a \prec_\uparrow b$ and $a \preceq_\uparrow b$ to refer to this order. We say that a vector $x$ in a set $D$ is lexicographically maximum if for all $y \in D$ we have $y \preceq_\uparrow x$.

Fujishige proved the following theorem for polymatroids.

▶ **Theorem 14** ([18])**.** *Let $f : 2^V \to \mathbb{R}_+$ be a monotone submodular function (a polymatroid) and let $B_f$ be its base polytope. Then there is a unique lexicographically maximum base $b^* \in B_f$ and for each $v \in V$, $b^*_v = \lambda_v$. Moroever, $b^*$ is the optimum solution to the quadratic program: $\min \sum_v x_v^2$ subject to $x \in B_f$.*

Another ordering is to sort the coordinates in decreasing order of value and then taking the lexicographic ordering on the two sorted sequences. We denote this ordering by $\prec_\downarrow, \preceq_\downarrow$ for strict and non-strict ordering respectively. We say that a vector $x$ in a set $D$ is lexicographically minimum if for all $y \in D$ we have $x \preceq_\downarrow y$. The preceding theorem can be generalized to contrapolymatroids in a straight forward fashion and this was explicitly pointed out in [24]. We paraphrase it to be similar to the preceding theorem statement.

▶ **Theorem 15.** *Let $f : 2^V \to \mathbb{R}_+$ be a monotone supermodular function (a contrapolymatroid) and let $B_f$ be its base polytope. Then there is a unique lexicographically minimum base $b^* \in B_f$ and for each $v \in V$, $b^*_v = \lambda_v$. Moreover, $b^*$ is the optimum solution to the quadratic program: $\min \sum_v x_v^2$ subject to $x \in B_f$.*

## 3.3 Approximating a lexicographically optimal base using Frank-Wolfe

Consider the convex quadratic program $\min \sum_{v \in V} x_v^2$ subject to $x \in B_f$ where $B_f$ is the base polytope of $f$ (could be submodular of supermodular). We can use the Frank-Wolfe method to approximately solve this optimization problem. The gradient of the quadratic function is $2x$ and it follows that in each iteration, we need to answer the linear minimization oracle of $\text{LMO}(w) = \arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, 2\mathbf{w} \rangle$ for $\mathbf{w} \in B_f$. This is equivalent to $\arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, \mathbf{w} \rangle$, in other words optimizing a linear objective over the base polytope. Edmonds [15] showed that the simple greedy algorithm is an $O(|V| \log |V|)$ time exact algorithm (assuming $O(1)$ time oracle access to $f$).

▶ **Theorem 16** ([15])**.** *Fix a polymatroid $f : 2^V \to \Re_+$. Given a weight vector $\mathbf{w} \in \Re^n$, let $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ be a sort of $V = \{v_1, ..., v_n\}$ in ascending order of $\mathbf{w}_i$ values. Let $A_i = \{v_{j_1}, ..., v_{j_i}\}$ for $1 \leq i \leq n$ with $A_0 = \emptyset$. Define $\mathbf{s}^*_i = f(A_i) - f(A_{i-1})$. Then $\mathbf{s}^* = \arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, \mathbf{w} \rangle$.*

The theorem also holds for supermodular functions but by reversing the order from ascending to descending order of $\mathbf{w}$ and complimenting the set $A_i$.

▶ **Theorem 17** ([15])**.** *Fix a contrapolymatroid $f : 2^V \to \Re_+$. Given a weight vector $\mathbf{w} \in \Re^n$, let $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ be a sort of $V = \{v_1, ..., v_n\}$ in descending order of $\mathbf{w}_i$ values. Let $A_i = \{v_{j_i}, ..., v_{j_n}\}$ for $1 \leq i \leq n$ with $A_{n+1} = \emptyset$. Define $\mathbf{s}^*_i = f(A_i) - f(A_{i+1})$. Then $\mathbf{s}^* = \arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, \mathbf{w} \rangle$.*

Both algorithms are dominated by the sorting step and thus takes $O(|V| \log |V|)$ time. These simple algorithms imply that the Frank-Wolfe algorithm can be used on the quadratic program to obtain an approximation to the lexicographically maximum (respectively minimum) bases of submodular (respectively supermodular) functions. The standard Frank-Wolfe algorithm would need $O(\frac{\text{diam}(B_f)^2}{\epsilon^2})$ iterations to converge to a vector $\hat{b}$ satisfying $\left\| \hat{b} - b^* \right\|_2 \leq \epsilon$.

# 4 Application 1: Convergence of Greedy++ and SuperGreedy++

We begin by describing Greedy++ [7] and its generlization SuperGreedy++ [10]. Greedy++ is built upon the peeling idea of Greedy, and applies it over several iterations. The algorithm initializes a weight/load on each $v \in V$, denoted by $w(v)$, to 0. In each iteration it creates an ordering by peeling the vertices: the next vertex to be chosen is $\arg\min_{v \in V(G')}(w(v) + \deg_{G'}(v))$ where $G'$ is the current graph (after removing the previously peeled vertices). At the end of the iteration, $w(v)$ is increased by the degree of $v$ when it was peeled in the current iteration. A precise description can be found below. SuperGreedy is a natural generalization of Greedy, and SuperGreedy++ generalizes Greedy++. A formal description of SuperGreedy++ is given below.

**Algorithm 2** GREEDY++$(G(V,E),T)$ [7].

Initialize $w(u) \leftarrow 0$ for all $u \in V$
$G^* \leftarrow G$
**for** $k \leftarrow 0$ to $T-1$ **do**
  $G' \leftarrow G$
  **while** $|G'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in G'}(w(u) + deg_{G'}(u))$
    $w(u) \leftarrow w(u) + deg_{G'}(u)$
    $G' \leftarrow G' - \{u\}$
    **if** $\lambda(G') > \lambda(G^*)$ **then**
      $G^* \leftarrow G'$
**return** $G^*$

**Algorithm 3** SUPER-GREEDY++$(f,T)$ [10].

Initialize $w(u) \leftarrow 0$ for all $u \in V$
$S^* \leftarrow V$
**for** $k \leftarrow 0$ to $T-1$ **do**
  $V' \leftarrow V$
  **while** $|V'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in V'}(w(u) + f(V') - f(V'-u))$
    $w(u) \leftarrow w(u) + f(V') - f(V'-u)$
    $V' \leftarrow V' - u$
    **if** $\frac{f(V')}{|V'|} > \frac{f(S^*)}{|S^*|}$ **then**
      $S^* \leftarrow V'$
**return** $S^*$

The goal of this section is to prove Theorem 1 on the convergence of SUPERGREEDY++ and GREEDY++ to the lexicographically maximal base.

## 4.1 Intuition and main technical lemmas

As we saw in Section 3.3, if one applies the Frank-Wolfe algorithm to solve the qaudratic program $\min \sum_{v \in V} x_v^2$ subject to $x \in B_f$, each iteration corresponds to finding a minimum weight base of $f$ where the weights are given by the current vector $x$. Finding a minimum weight base corresponds to sorting $V$ by $x$. However, SUPERGREEDY++ and GREEDY++ use a more involved peeling algorithm in each iteration; the peeling is based on the weights as well as the degrees of the vertices and it is not a static ordering (the degrees change as peeling proceeds). This is the difficulty in formally analyzing these algorithms. In [10], the authors used a connection to the multiplicative weight update method via LP relaxations. Here we rely on the quadratic program and noisy Frank-Wolfe. The high-level intuition, that originates in [10], is the following. As the algorithm proceeds in iterations, the weights on the vertices accumulate; recall that the total increase in the weight in the case of DSG is $m = |E|$. The degree term, which influences the peeling, is dominant in early iterations, but its influence on the ordering of the vertices decreases eventually as the weights of the vertices get larger. It is then plausible to conjecture that the algorithm behaves like the standard Frank-Wolfe method in the limit. The main question is how to make this intuition precise. [10] relies on a connection to the MWU method while we use a connection to noisy Frank-Wolfe.

For this purpose, consider an iteration of GREEDY++ and SUPERGREEDY++. The algorithm peels based on the current weight vector and the degrees. We isolate and abstract this peeling algorithm and refer to it as Weighted-Greedy and Weighted-SuperGreedy respectively, and formally describe them with the weight vector $w$ as a parameter.

**Algorithm 4** WEIGHTED-GREEDY($G$, $w$).

Input: $G(V, E)$ and $w(u)$ for $u \in V$
  $G' \leftarrow G$
  Initialize $\hat{d}(u) = 0$ for all $u \in V$.
  **while** $|G'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in G'}(w(u) + deg_{G'}(u))$
    $\hat{d}(u) \leftarrow deg_{G'}(u)$
    $G' \leftarrow G' - \{u\}$
  **return** $\hat{d}$

**Algorithm 5** WEIGHTED-SUPERGREEDY($f$, $w$).

Input: Supermodular $f : 2^V \to \Re_+$, $w(u)$ for $u \in V$
  $V' \leftarrow V$
  Initialize $\hat{d}(u) = 0$ for all $u \in V$.
  **while** $|V'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in G'}(w(u) + f(V') - f(V' - u)$
    $\hat{d}(u) \leftarrow f(V') - f(V' - u)$
    $V' \leftarrow V' - u$
  **return** $\hat{d}$

The peeling algorithms also compute a base $\hat{d} \in B_f$. In the case of graphs and DSG, $\hat{d}(u)$ is set to the degree of the vertex $u$ when it is peeled. One can alternatively view the base as an orientation of the edges of $E$. Define for each edge $uv \in G$ two weights $x_{uv}, x_{vu}$. We say that $\mathbf{x}$ is *valid* if $x_{uv} + x_{vu} = 1$ and $x_{uv}, x_{vu} \geq 0$ for all $\{u, v\} \in E(G)$. For $b \in \Re^{|V|}$, we say $x$ *induces* $b$ if $b_u = \sum_{v \in \delta(u)} x_{uv}$ for all $u \in V$. We say a vector $d$ is an *orientation* if there is a valid $x$ that induces it.

▶ **Lemma 18** ([24]). *For $f(S) = |E(S)|$, $b \in B_f$ if and only if $b$ is an orientation.*

Recall that the Frank-Wolfe algorithm, for a given weight vector $w : V \to \mathbb{R}_+$, computes the minimum-weight base $b$ with respect to $w$ since $\langle w, b\rangle = \min_{y \in B_f} \langle w, y\rangle$. It is worth taking a moment to note that this base (or orientation due to Lemma 18) is easily computable: we orient each edge integrally (i.e $x_{vu} = 1, x_{uv} = 0$) from $v$ to $u$ if $w(u) \geq w(v)$, and from $u$ to $v$ otherwise. A simple exchange argument yields a proof of correctness and is implicit in many works [14][2]. This induces an optimal base $d^*_w$ with respect to $w$. Our goal is to compare how the peeling order created by Weighted-Greedy (and Weighted-SuperGreedy) compares with the best base. The following two technical lemmas formalize the key idea. The first is tailored to DSG and the second applies to DSS.

▶ **Lemma 19.** *Let $\hat{d}$ be the output from WEIGHTED-GREEDY$(G, w)$ and $d^*_w$ be the optimal orientation with respect to $w$. Then $\langle w, \hat{d}\rangle \leq \langle w, d^*_w\rangle + \sum_u deg_G(u)^2$. In particular, the additive error **does not depend** on the weight vector $w$.*

▶ **Lemma 20.** *For a supermodular function $f : 2^V \to \Re_+$, let $\hat{d}$ be the output from WEIGHTED-SUPERGREEDY$(f, w)$ (Algorithm 5) and $d^*_w$ be the optimal vector with respect to $w$ as described in Theorem 17. Then $\langle w, \hat{d}\rangle \leq \langle w, d^*_w\rangle + n\sum_{u \in V} f(u \mid V - u)^2$. In particular, the additive error **does not depend** on the weight vector $w$.*

## 4.2 Convergence proof for Greedy++

Why is Lemma 19 crucial? First, observe that the minimizer $d^*_w$ of $\langle w, d\rangle$ is exactly the same minimizer as $\langle Kw, d\rangle$ for any constant $K > 0$ (and vice-versa).

▶ **Lemma 21.** *Let $\hat{d}_K$ be the output of WEIGHTED-GREEDY$(G, Kw)$. Then $\langle w, \hat{d}_K\rangle \leq \langle w, d^*_w\rangle + \frac{\sum_u deg_G(u)^2}{K}$.*

[2] Since the optimal orientation is easily computable, one can replace the "peeling" iteration of GREEDY++ with the optimum base. This would result in the Frank-Wolfe based algorithm of [14].

**Proof.** By Lemma 19, $\sum_{u\in V} Kw(u)\hat{d}_K(u) \leq \min_{\text{orientation } d} \left(\sum_{u\in V} Kw(u)d(u)\right) + \sum_u deg_G(u)^2$. Dividing by $K$ implies the claim. ◀

We are now ready to view GREEDY++ as a noisy Frank-Wolfe algorithm. Algorithm 6 shows how GREEDY++ could be interpreted.

**Algorithm 6** GREEDY++$(G(V,E))$.

Input: $G=(V,E)$ and $w(u)$ for $u \in V$
  Initialize $b^{(0)} \leftarrow$ WEIGHTED-GREEDY$(G, \mathbf{0})$ ▷ $b^{(0)}$ is a valid orientation
  **for** $k \leftarrow 0$ to $T-1$ **do**
    $\gamma \leftarrow \frac{1}{k+1}$
    $d^{(k+1)} \leftarrow$ WEIGHTED-GREEDY$(G, (k+1)b^{(k)})$
    $b^{(k+1)} \leftarrow (1-\gamma)b^{(k)} + \gamma d^{(k+1)}$
  **return** $b^{(T)}$

The algorithm is exactly the same as the one described in Algorithm 2. Indeed, one can prove that $kb^{(k)}$ is precisely the weights that GREEDY++ ends with at round $k$ by induction. Observe that $(k+1)b^{(k+1)} = kb^{(k)} + d^{(k+1)}$ which is precisely the load as described in Algorithm 2 (via induction). We note that $\gamma \leftarrow 1/(k+1)$ is crucial here to ensure we are taking the average. Lemma 25 in the appendix (full version) implies that each peel in Algorithm 2 is $\frac{\delta C_f}{k+2}$-approximate linear minimization oracle. Using Theorem 8, this implies that GREEDY++ (as described in Algorithm 2) converges to $b^*$ in $\tilde{O}(\frac{mn^2}{\epsilon^2})$ iterations since $\delta = O(\frac{\sum_u d_G(u)^2}{m})$ and $C_f = O(\sum_u d_G(u)^2)$. We use the probabilistic method to bound $C_f$ in the full version.

**Extension to SuperGreedy++.** An essentially similar analysis works for SUPERGREEDY++. Instead of Lemma 19, we rely on Lemma 20. For technical reasons, the convergence analysis of SUPERGREEDY++ is slightly weaker than for GREEDY++.

## 5 Application 2: Greedy Tree Packing interpreted via Frank-Wolfe

Let $G=(V,E)$ be a graph with non-negative edge capacities. The goal of this section is to view Thorup's definitions of ideal edge loads and the associated tree packing from a different perspective, and to derive an alternate convergence analysis of his greedy tree packing algorithm [46, 47]. In previous work, Chekuri, Quanrud and Xu [11] obtained a different tree packing based on an LP relaxation for $k$-cut, and used it in place of ideal tree packing. Despite this, there was a gap in our understanding which we address here.

We restrict our attention to unweighted multi-graphs throughout this section, and comment on the capacitated case at the end of the section. Let $G=(V,E)$ be a connected multi-graph, with $n$ vertices and $m$ edges. Consider the graphic matroid $\mathcal{M}_G(E,\mathcal{F})$ induced by $G$; $E$ is the ground set, and $\mathcal{F}$ consists of all sub-forests of $G$. The bases of the matroid are precisely the spanning trees of $G$. Consider the rank function $r: 2^E \to \mathbb{Z}_+$ of $\mathcal{M}_G$. $r$ is submodular, and it is well-known that for a edge subset $X \subseteq E$, $r(X) = n - \kappa(X)$ where $\kappa(X)$ is the number of connected components induced by $X$.

## 5.1 Thorup's recursive algorithm as dense decomposition

For consistency with previous notation, we use $f$ to denote the submodular rank function $r$. We first describe ideal loads as defined by Thorup. Consider the Tutte–Nash-Williams partition $P$ for $G$. Recall that $P$ minimizes the ratio $\frac{|E(P)|}{|P|-1}$ among all partitions, and this ratio is $\tau(G)$. For each edge $e \in E(P)$, assign $\ell^*(e) = \frac{1}{\tau(G)}$. Remove the edges in $E(P)$ to obtain a graph $G'$ which now consists of several disconnected components. Recursively compute ideal loads for the edges in each connected component of $G'$ (the process stops when $G$ has no edges).

We claim that Thorup's recursive decomposition coincides with the dense decomposition of $f$ (the first variant). To see this, it suffices to see the first step of the dense decomposition. We find the minimal set $S_1 \subseteq E$ that minimizes $\frac{|E|-|S|}{f(E)-f(S)}$. We let $\hat{S}_1 = E \setminus S_1$ and assign the edges in $\hat{S}_1$ the density $\frac{f(E)-f(S)}{|E|-|S|}$. Then, we "delete" $\hat{S}_1$. Observe that $\hat{S}_1 = E \setminus S_1$ is just the edges crossing the partition $P(S_1)$ defined by the $\kappa(S_1)$ connected components spanned by $S_1$. Also, recall that $\frac{f(E)-f(S_1)}{|E|-|S_1|} = \frac{\kappa(S_1)-1}{|E\setminus S_1|} = \frac{|P(S_1)|-1}{E(P(S_1))} = \frac{1}{\tau(G)}$. Hence, the density assigned to edges in $\hat{S}_1$ is exactly $\frac{1}{\tau(G)}$ by the Tutte–Nash-Williams theorem. The next step is deleting $\hat{S}_1 = E \setminus S_1$, which, as discussed above, are the edges crossing the partition $P(S_1)$.

Via induction we prove the following lemma.

▶ **Lemma 22.** *The weights given to the edges by the dense decomposition algorithm on $f$ coincide with $\ell^*$.*

## 5.2 Greedy tree packing converge to ideal relative loads

Thorup considered the following greedy tree packing algorithm. For each edge define a load $\ell(e)$ which is initialized to 0. The algorithm proceeds in iterations. In iteration $i$ the algorithm computes an MST $T_i$ in $G$ with respect to edge weights $w(e) = \ell(e)$. The load of each edge $e \in T_i$ is increased by 1. Thorup showed that as $k \to \infty$, the quantity $\ell(e)/k$ converges to $\ell^*(e)$ for each edge $e$. His proof is fairly technical. In this section, we present a different proof of this fact that uses the machinery we have built thus far.

▶ **Lemma 23.** *The vector $\ell^*$ is the lexicographically maximal base of the spanning tree polytope.*

**Proof.** We showed that Thorup's definition of ideal loads is obtained by simply running the dense decomposition on the rank function of the graphic matroid induced by $G$. The bases of the graphic matroid are the spanning trees of $G$ and hence the base polytope of $f$ is the spanning tree polytope of $G$. The dense decomposition of $f$ gives us the lexicographically maximum base, and hence $\ell^*$ is the lexicographically maximal base in the spanning tree polytope of $G$. ◀

Hence, $\ell^*$ is the unique solution to the quadratic program: min $\sum_e \ell(e)^2$ subject to $\ell \in \mathrm{SPT}(G)$ where $\mathrm{SPT}(G)$ is the spanning tree base polytope. We can thus apply a noisy Frank-Wolfe algorithm to the quadratic program to obtain Algorithm 7.

The main observation is that this algorithm is **exactly** the same as Thorup's greedy tree packing algorithm. Indeed, observe that $(k+1)\ell^{(k+1)} \leftarrow k\ell^{(k)} + d^{(k+1)} = k\ell^{(k)} + \mathbb{1}\{e \in \mathrm{MST}(G, \ell^{(k)})\}$ where $\mathrm{MST}(G, w)$ is a minimum spanning tree of $G$ with respect to edge weights $w$. Since noisy Frank-Wolfe converges, then $\ell^{(k)}$ converges to $\ell^*(e)$, and greedy tree packing converges.

**Algorithm 7** FRANK-WOLFE-GREEDY-TREEPACK($G(V, E)$).

Input: $G(V, E)$
  Initialize $l^{(0)}(u) = \mathbb{1}\{e \in T\}$ for any spanning tree $T$.
  **for** $k \leftarrow 0$ to $T - 1$ **do**
    $\gamma \leftarrow \frac{1}{k+1}$
    $d^{(k+1)} \leftarrow \min_{s \in \mathrm{SPT}(G)} \langle l^{(k)}, s \rangle$ ▷ This is the minimum spanning tree with respect to $l^{(k)}$
    $l^{(k+1)} \leftarrow (1-\gamma) l^{(k)} + \gamma d^{(k+1)}$
  **return** $b^{(T)}$

We now establish the convergence guarantee for greedy tree packing. For the spanning tree polytope of an $m$ edge graph, the curvature constant $C_f \leq 4m$ because for $x, y \in B_f$, $2(x-y)^T(x-y) = \sum_{e \in E}(x_e - y_e)^2 \leq 4m$. Plugging this bound into Theorem 8, after $k = O(\frac{m \log(m/\epsilon)}{\epsilon^2})$ iterations, $\left\|\ell^{(k)} - \ell^*\right\|_2 \leq \epsilon$.

Suppose we run the standard Frank-Wolfe algorithm with $\gamma = 2/(k+2)$. Then, the convergence guarantee improves to $O(\frac{m}{\epsilon^2})$. Note that each iteration still corresponds to finding an MST in the graph with weights. However, the load vector is no longer a simple average of the trees taken so far.

**Comparison to Thorup's bound and analysis.** Thorup [47] considered ideal tree packings in capacitated graphs; let $c(e) \geq 1$ (via scaling) denote the capacity of edge $e$. Via [18], one sees that the optimum solution of the quadratic program $\sum_e x_e^2/c(e)$ subject to $x \in SP(G)$ is the ideal load vector $\ell^*$. Greedy tree packing generalizes to the capacitated case easily; in each iteration we compute the MST with respect to weights $w(e) = \ell(e)c(e)$. Thorup proved the following.

▶ **Theorem 24** ([47]). *Let $G = (V, E)$ be capacitated graph. Greedy tree packing after $O(\frac{m \log(mn/\epsilon)}{\epsilon^3})$ iterations ouputs a load vector $\ell$ such that for each edge $e \in E$, $\ell(e) \leq (1+\epsilon)\ell^*(e)$.*

We observe that if all capacities are 1 (or identical) then Thorup's guarantee is that $\ell(e) - \ell^*(e) \leq O(\epsilon)$ for each edge $e$. For this case, via Frank-Wolfe, we obtain the much stronger guarantee that $||\ell - \ell^*||_2 \leq \epsilon$ which easily implies the per edge condition, however the per edge guarantee does not imply a guarantee on the norm. Further, in the unweighted case, our iteration complexity dependence on $\epsilon$ is $1/\epsilon^2$ while Thorup's is $1/\epsilon^3$. Thorup's guarantee works for the capacitated case in strongly polynomial number of iterations. We can adapt the Frank-Wolfe analysis to the capacitated case but it would yield a bound that depends on $C = \sum_e c(e)$ (in the unweighted case $C = m$); on the other hand the guarantee provided by Frank-Wolfe is stronger.

It may seem surprising that the same greedy tree packing algorithm yields different types of guarantees based on the type of analysis used. We do not have a completely satisfactory explanation but we point out the following. Thorup's analysis is a non-trivial refinement of the standard MWU type analysis of tree packing [38, 50, 3]. As already noted in [24], if one uses Frank-Wolfe (with $\gamma = 1/(k+1)$) with the softmax potential function that is standard in the MWU framework, then the resulting algorithm would also be greedy tree packing. Fujishige's uses a quadratic objective to guarantee that the optimum solution is the unique maximal base but in fact any increasing strongly convex function would suffice. In the context of optimizing a linear function over $B_f$, due to the optimality of the greedy algorithm, the only thing that determines the base is the ordering of the elements of $V$ according to the weight vector; the weights themselves do not matter. Thus, Frank-Wolfe applied to different

convex objectives can result in the same greedy tree/base packing algorithm. However, the specific objective can determine the guarantee one obtains after a number of iterations. The softmax objective is better suited for obtaining relative error guarantees while the quadratic objective is better suited for obtaining additive error guarantees. Thorup's analysis is more sophisticated due to the per edge guarantee in the capacitated setting. A unified analysis that explains both the relative and additive guarantees is desirable. We leave this is an interesting direction for future research.

## References


1. Reid Andersen and Kumar Chellapilla. Finding dense subgraphs with size bounds. In Konstantin Avrachenkov, Debora Donato, and Nelly Litvak, editors, *Algorithms and Models for the Web-Graph*, pages 25–37, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
2. Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, February 2012. doi:10.14778/2168651.2168658.
3. Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012.
4. Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.
5. Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for mapreduce. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–78. Springer, 2014.
6. Oana Denisa Balalau, Francesco Bonchi, T-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 379–388, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2684822.2685298.
7. Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. *Flowless: Extracting Densest Subgraphs Without Flow Computations*, pages 573–583. Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3366423.3380140.
8. Digvijay Boob, Saurabh Sawlani, and Di Wang. Faster width-dependent algorithm for mixed packing and covering lps. *Advances in Neural Information Processing Systems 32 (NIPS 2019)*, 2019.
9. Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, pages 84–95, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
10. Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1531–1555, 2022. doi:10.1137/1.9781611977073.64.
11. Chandra Chekuri, Kent Quanrud, and Chao Xu. Lp relaxation and tree packing for minimum k-cut. *SIAM Journal on Discrete Mathematics*, 34(2):1334–1353, 2020.
12. Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time, 2022. arXiv:2203.00671.
13. Aleksander B. G. Christiansen, Jacob Holm, Ivor van der Hoog, Eva Rotenberg, and Chris Schwiegelshohn. Adaptive out-orientations with applications, 2023. arXiv:2209.14087.
14. Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 233–242, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. doi:10.1145/3038912.3052619.

15 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications (Proceedings Calgary International Conference on Combinatorial Structures and Their Applications, Calgary, Alberta, 1969; R. Guy, H. Hanani, N. Sauer, J. Schönheim, eds.)*, New York, 1970. Gordon and Breach.

16 Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 300–310, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. doi:10.1145/2736277.2741638.

17 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956. doi:10.1002/nav.3800030109.

18 Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980. URL: http://www.jstor.org/stable/3689149.

19 Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.

20 Satoru Fujishige. Theory of principal partitions revisited. *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 127–162, 2009.

21 Takuro Fukunaga. Computing minimum multiway cuts in hypergraphs from hypertree packings. In *IPCO*, pages 15–28. Springer, 2010.

22 A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5956.html.

23 Anupam Gupta, David G Harris, Euiwoong Lee, and Jason Li. Optimal bounds for the k-cut problem. *ACM Journal of the ACM (JACM)*, 69(1):1–18, 2021.

24 Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. Faster and scalable algorithms for densest subgraph and decomposition. In *Advances in Neural Information Processing Systems*, volume 35, pages 26966–26979. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/ac8fbba029dadca99d6b8c3f913d3ed6-Paper-Conference.pdf.

25 Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, number 1 in Proceedings of Machine Learning Research, pages 427–435, Atlanta, Georgia, USA, 17–19 June 2013. PMLR. URL: https://proceedings.mlr.press/v28/jaggi13.html.

26 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000.

27 Yuko Kuroki, Atsushi Miyauchi, Junya Honda, and Masashi Sugiyama. Online dense subgraph discovery via blurred-graph feedback. In *ICML*, 2020.

28 Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants, 2023. arXiv:2303.14467.

29 Jason Li. Faster minimum k-cut of a simple graph. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1056–1077. IEEE, 2019.

30 Xiangfeng Li, Shenghua Liu, Zifeng Li, Xiaotian Han, Chuan Shi, Bryan Hooi, He Huang, and Xueqi Cheng. Flowscope: Spotting money laundering based on graphs. In *AAAI*, 2020.

31 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-cut. *SIAM Journal on Computing*, 0:FOCS20–205, 2022.

32 Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V.S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pages 1051–1066, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3318464.3389697.

**33** Fragkiskos D Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, and Michalis Vazirgiannis. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal*, 29(1):61–92, 2020.

**34** Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Mathematical Foundations of Computer Science 2015*, pages 472–482, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

**35** H Narayanan. The principal lattice of partitions of a submodular function. *Linear Algebra and its Applications*, 144:179–216, 1991.

**36** Hariharan Narayanan. *Submodular functions and electrical networks*, volume 54. Elsevier, 1997.

**37** Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982.

**38** Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. URL: `http://www.jstor.org/stable/3690406`.

**39** Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. Discovering dynamic communities in interaction networks. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 678–693, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

**40** Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 181–193. ACM, 2020. `doi:10.1145/3357713.3384327`.

**41** Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

**42** Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Corescope: Graph mining using k-core analysis — patterns, anomalies and algorithms. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 469–478, 2016. `doi:10.1109/ICDM.2016.0058`.

**43** Bintao Sun, Maximilien Danisch, T-H. Hubert Chan, and Mauro Sozio. Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proc. VLDB Endow.*, 13(10):1628–1640, June 2020. `doi:10.14778/3401960.3401962`.

**44** Nikolaj Tatti. Density-friendly graph decomposition. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(5):1–29, 2019.

**45** Nikolaj Tatti and Aristides Gionis. Density-friendly graph decomposition. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1089–1099, 2015.

**46** Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. Preliminary version in Proc. of ACM STOC 2001.

**47** Mikkel Thorup. Minimum k-way cuts via deterministic greedy tree packing. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 159–166, 2008.

**48** Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1122–1132, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. `doi:10.1145/2736277.2741098`.

**49** Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 104–112, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2487575.2487645`.

**50** N. Young. Randomized rounding without solving the linear program. In *ACM-SIAM Symposium on Discrete Algorithms*, 1995.

# Algorithms for Matrix Multiplication via Sampling and Opportunistic Matrix Multiplication

**David G. Harris**
University of Maryland, College Park, MD, USA

## Abstract

Karppa & Kaski (2019) proposed a novel type of "broken" or "opportunistic" multiplication algorithm, based on a variant of Strassen's algorithm, and used this to develop new algorithms for Boolean matrix multiplication, among other tasks. For instance, their algorithm can compute Boolean matrix multiplication in $O(n^{\log_2(6+6/7)} \log n) = O(n^{2.778})$ time. While faster matrix multiplication algorithms exist asymptotically, in practice most such algorithms are infeasible for practical problems.

We describe an alternative way to use the broken matrix multiplication algorithm to approximately compute matrix multiplication, either for real-valued matrices or Boolean matrices. In brief, instead of running multiple iterations of the broken algorithm on the original input matrix, we form a new larger matrix by sampling and run a single iteration of the broken algorithm. Asymptotically, the resulting algorithm has runtime $O(n^{\frac{3 \log 6}{\log 7}} \log n) \leq O(n^{2.763})$, a slight improvement of Karppa-Kaski's algorithm.

Since the goal is to obtain new practical matrix-multiplication algorithms, these asymptotic runtime bounds are not directly useful. We estimate the runtime for our algorithm for some sample problems which are at the upper limits of practical algorithms; it appears that for these parameters, further optimizations are still needed to make our algorithm competitive.

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

**Keywords and phrases** Matrix multiplication, Boolean matrix multiplication, Strassen's algorithmkeyword

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.57

**Related Version** *Full Version*: https://arxiv.org/abs/2109.13335

**Acknowledgements** Thanks for Richard Stong for suggesting the proof of Lemma 10.

## 1 Introduction

Consider the fundamental computational problem of *matrix multiplication*: given matrices $A, B$ of dimensions $d_1 \times d_3$ and $d_3 \times d_2$ respectively, our goal is to compute the matrix $C$ given by

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

When the matrices are square, we write $n = d_1 = d_2 = d_3$. There is an obvious $O(d_1 d_2 d_3)$ algorithm (the so-called "naive algorithm"), which simply iterates over all values $i, j, k$. There is a long line of research into a variety of asymptotically faster algorithms. For square matrices, the runtime is given as $n^{\omega+o(1)}$, where $\omega$ is the linear algebra constant. Currently, the best bound [2] is $\omega \leq 2.38$, coming from a variant of Coppersmith-Winograd's algorithm [6]. There are also fast algorithms for rectangular matrix multiplication, although they are less heavily studied [12, 7].

Unfortunately, nearly all the fast matrix multiplications algorithms are completely impractical due to large hidden constants. There are a small handful of algorithms which are efficient *in practice*; by far the most important is Strassen's algorithms and its variants,


which have runtime $O(n^{\log_2 7})$. Depending on the matrix shape, a few rectangular algorithms may also be practical [5]. There is an extensive literature on optimizing and implementing Strassen's algorithm in various computational platforms, see e.g [4, 8, 10].

There is an important special case of *Boolean* matrix multiplication (BMM). Here, the entries of $A, B$ come from the algebra $\{0, 1\}$ with binary operations $\vee, \cdot$, and our goal is to compute the matrix $C$ given by

$$C_{ij} = \bigvee_k A_{ik} B_{kj}$$

This problem, along with some variants, is a primitive used in algorithms for transitive closures, parsing context-free grammars, and shortest path problems in unweighted graphs, among other applications. Again, the obvious $O(d_1 d_2 d_3)$ naive algorithm can be used. There are a number of other specialized algorithms based on combinatorial optimizations; most recently, [13] described an algorithm with runtime roughbly $O(n^3/\log^4 n)$.

There is a standard reduction from BMM to integer matrix multiplication: compute the matrix product $\tilde{C} = AB$ over the integers, and then set $C_{ij} = 1$ if $\tilde{C}_{ij} \geq 1$. Alternatively, there is a randomized reduction from BMM to matrix multiplication over the finite field GF(2): each non-zero entry of $B$ is set to zero with probability 1/2, and then we compute the matrix product $\tilde{C} = AB$ over GF(2). Then $\tilde{C}_{ij} = 1$ with probability 1/2 whenever $C_{ij} = 1$, else $\tilde{C}_{ij} = 0$ with probability one. With further $O(\log n)$ repetitions the error probability can be reduced to a negligible level. The advantage of this approach is that a variety of additional optimizations are possible for GF(2) arithmetic, most importantly, the Four Russians method [3]. These can often be combined with asymptotically fast algorithms such as Strassen. For example, [1] provides an optimized bitsliced implementation which uses Strassen for the high-level iterations.

It seems difficult to make progress on better practical algorithms for matrix multiplication. In [11], Karppa & Kaski proposed an innovative and novel approach to break this impasse: they described a "broken" or "opportunistic" form of matrix multiplication, which computes the matrix product with some high failure probability. This broken multiplication, in turn, can be computed more efficiently, both asymptotically and practically, by a variant of Strassen's algorithm. For brevity, we refer to their algorithm as the *KK algorithm*.

By iterating for sufficiently many repetitions, this procedure an be used to solve BMM with high probability. With appropriate choice of parameters, the overall runtime is $O(n^{\log_2(6+6/7)} \log n) \approx n^{2.776}$, a notable improvement over Strassen. See also [10] for further details.

In this paper, we describe an alternative way to use the KK algorithm: instead of executing multiple independent iterations, we combine them all into a single larger randomized broken multiplication. Each term of the original matrix multiplication product gets sampled multiple times into the larger matrix.

For Boolean matrix multiplication, this gives the following crisp result (given here in a slightly simplified form):

▶ **Theorem 1.** *For square matrices, we can compute BMM using* $O(n^{\frac{3\log 3}{\log 7}} (\log n)^{\frac{\log 6}{\log 7}}) \leq O(n^{2.763})$ *bit operations with high probability*

Our algorithm can also be used to obtain a randomized approximation for real-valued matrix multiplication. The precise approximation guarantees depend on the values input matrices. For general matrices, the accuracy of the estimated matrix term $\tilde{C}_{ij}$ depends on the value $C_{ij}$ as well as the "second moment" of matrix, namely

$$\tilde{C}_{ij}^{(2)} = \sum_k A_{ik}^2 B_{kj}^2$$

When the input matrices $A, B$ (and hence $C$) are non-negative, then note that $\tilde{C}_{ij}^{(2)} \leq C_{ij}^2$; in this case, we have the following crisp statement (again, in simplified form):

▶ **Theorem 2.** *For non-negative square matrices of dimension $n$, we can obtain an approximating matrix $\tilde{C}$ using $O(n^{2.763}/\varepsilon^2)$ operations with high probability, satisfying $(1-\epsilon)C_{ij} \leq \tilde{C}_{ij} \leq (1+\epsilon)C_{ij}$ for all $i, j$.*

For example, in the setting of Boolean matrix multiplication, we can not only *detect* if there is any value $k$ with $A_{ik} = B_{kj} = 1$, but we can *estimate* the number of such values of $k$, up to any desired relative accuracy.

The new algorithm is quite simple, and also has a number of advantages from the viewpoint of practical implementations. First, it lends itself easily to rectangular (non-square) input matrices; we will show a more general theorem which describes the runtime scaling for matrices of arbitrary shape. Second, the algorithm itself internally uses square matrices, and is flexible about the precise dimensions used; this avoids a number of tedious issues with padding in matrix algorithms.

We emphasize that this paper is structured somewhat differently from a typical paper in theoretical computer science. Since our goal is to develop a new *practical* algorithm (we already have good theoretical algorithms!), we cannot afford to look only at asymptotic estimates, ignoring constant factors in runtime. Instead, we have developed our formlas and calculations much more preicsely to get concrete runtime estimates in addition to the usual asymptotic bounds.

In Section 7, we illustrate with some computational estimates for large problem instances which are at the limit of practicality. Unfortunately, as we will see, the asymptotic behavior does not seem to fully kick in even at such large scales. At the current time, it appears that the new algorithm is unlikely to beat alternative algorithms; further optimizations and improvements may be needed.

We note that [11] also discussed generalizations to other types of broken matrix multiplication tensors. We will restrict our attention of the Strassen-based pseudo-multiplication, for three reasons. First, much of our analysis depends on certain structural properties of Strassen-type tensor; unlike the analysis in [11], which only depended on the gross *number of terms* in the pseudo-multiplication, we need to track the distribution of terms carefully. Second, it is not clear if any other tensors are practical, especially in light of the fact that Strassen's algorithm appears to be the only truly practical fast multiplication algorithm known. Finally, many of the "generic" amplification techniques for general tensors, which "only" lose constant factors, are too crude for understanding practical implementations.

## 2 Pseudo-multiplication

To describe our BMM algorithm, we need to discuss the underlying broken multiplication algorithm in depth. We call this procedure "pseudo-multiplication", to distinguish it from the overall KK algorithm itself. We begin by quoting the algorithmic result of [11].

▶ **Theorem 3.** *Consider* $2 \times 2$ *matrices* $A, B$ *over a ring* $R$. *Using* 14 *additions and* 6 *multiplications in* $R$, *we can compute the following quantities:*

$$C_{11} = A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

This formula differs from the computation of $C = AB$, in that the term $C_{11}$ is missing the summand $A_{11}B_{11}$. This computation is achieved by a variant of a Strassen step, except that one of the 7 multiplications is omitted.

Now consider integer matrices $A, B$ of dimension $n = 2^s$. We can identify the integers in the range $[n] = \{0, \dots, n-1\}$ with binary vectors $\{0,1\}^s$; thus, we may write $A_{xy}$ for vectors $x, y \in \{0,1\}^s$. We write $z = x \vee y$ where $z_i$ is zero iff $x_i = y_i = 0$, i.e. the disjunction is done coordinate-wise. Also, we let $|x|$ denote the Hamming weight of a binary vector $x$. By iterating the algorithm of Theorem 3 for $s$ levels, we get the following:

▶ **Theorem 4.** *Using* $7(6^s - 4^s)$ *additions and* $6^s = n^{\log_2 6}$ *multiplications, we can compute the matrix pseudo-product* $C = A \boxtimes B$ *defined as* $C_{xy} = \sum_{x \vee y \vee z = \vec{1}^s} A_{xz}B_{zy}$, *where* $\vec{1}^s$ *denotes the vector of dimension* $s$ *whose entries are all equal to* 1. *The computation of* $C$ *can be performed over any ring.*

**Proof.** See full paper. ◀

In particular, this implies the following result (which was the only thing shown directly in [11]):

▶ **Corollary 5.** *The pseudo-product* $C = A \boxtimes B$ *contains* $(7/8)^s$ *of the summands in the full matrix product* $C = AB$.

**Proof.** There are precisely $7^s$ triples of vectors $x, y, z \in \{0,1\}^s$ with $x \vee y \vee z = \vec{1}^s$. ◀

Note that [11] includes an additional randomization step, where the entries of the matrices are randomly permuted at each level. We omit this step, since we will later include more extensive randomization in the overall algorithm. (For practical purposes, this shuffling step can be awkward to implement efficiently). For given $s$, we define $T_s \subseteq \{0,1\}^s \times \{0,1\}^s \times \{0,1\}^s$ to be the set of triples $x, y, z$ with $x \vee y \vee z = \vec{1}^s$.

It is sometimes useful to use Strassen's algorithm, or naive matrix multiplication, for some of the low-level iterations. Let us suppose that we use $s$ steps of pseudo-multiplication, followed by true matrix multiplication on the resulting submatrices of dimension $b_i = d_i/2^s$ (the *base case*). We can view any integer $x \in \{1, \dots, d_i\}$ as equivalent to an ordered pair $(x', x'')$ where $x' \in \{0,1\}^s$ and $x'' \in \{1, \dots, b_i\}$. The resulting matrix, denoted still by $C = A \boxtimes B$, then satisfies

$$C_{xy} = \sum_{\substack{z=(z',z'') \in \{0,1\}^s \times \{1,\dots,b\} \\ x' \vee y' \vee z' = \vec{1}^s}} A_{(x',x''),(z',z'')} B_{(z',z'')(y',y'')}$$

The proof is completely analogous to Theorem 4 so we omit it here. For convenience, let us say that, for integers $x, y, z$, we have $x \vee y \vee z = \vec{1}$ if, when we write $x = (x', x''), y = (y', y''), z = (z', z'')$, we have $x' \vee y' \vee z' = \vec{1}^s$, i.e. $(x', y', z') \in T_s$. When $s, d_1, d_2, d_3$ are understood, we also define $T^* \subseteq [d_1] \times [d_2] \times [d_3]$ to be the resulting set of triples $(x, y, z)$ with this property. With these conventions, we can define $C = A \boxtimes B$ more compactly as $C_{xy} = \sum_{z:(x,y,z) \in T^*} A_{xz}B_{zy}$.

If the matrices are square, then the base case would also likely be square $b = b_1 = b_2 = b_3$. From a theoretical point of view, $b$ may be viewed as a constant. When working in the field GF(2), the base case multiplication would likely involve switching to a Four Russians method and/or bitsliced operations. These optimizations are very powerful in practice, possibly more important than the asymptotic gains from a fast matrix multiplication algorithm. Consequently, in practical implementations, $b$ may be very large; for instance, in the implementation of [1], they choose $b \approx 2^{11}$ (roughly matching the L2 cache size).

## 3 Algorithm overview

Consider matrices $A, B$ of dimension $d_1 \times d_3$ and $d_3 \times d_2$; here the matrices are either real-valued or Boolean-valued, and our goal is to approximate the product $C = AB$. The algorithms we use are very similar for these two cases: we choose some parameter $s$ and base sizes $b_1, b_2, b_3$. For $\ell = 1, 2, 3$ we define $m_\ell = 2^s b_\ell$, and draw random functions $f_\ell : [m_\ell] \to [d_\ell]$. We then form matrices $\bar{A}, \bar{B}$ of dimension $m_1 \times m_3, m_3 \times m_2$ by sampling from matrices $A, B$ according to the function $f$, i.e. we will have

$$\bar{A}_{xz} = A_{f_1(x) f_3(z)}, \qquad \bar{B}_{zy} = B_{f_3(z) f_2(y)}$$

(the actual matrices $\bar{A}, \bar{B}$ will have some additional scaling factors)

We then compute the pseudo-product $\bar{C} = \bar{A} \boxtimes \bar{B}$, with the given base case sizes, and produce an estimated matrix $\tilde{C}$, where, for each entry $ij$, the estimate $\tilde{C}_{ij}$ is derived by aggregating the entries $\bar{C}_{xy}$ where $x \in f_1^{-1}(i), y \in f_2^{-1}(j)$.

The runtime for this process will be $O(6^s)$ field operations, and the memory required is $O(4^s)$ to store the matrices $\bar{A}, \bar{B}, \bar{C}$ (assuming $b$ is constant). For any triple $i, j, k$ in the matrix product $A_{ik} B_{kj}$, there are roughly $\frac{7^s b_1 b_2 b_3}{d_1 d_2 d_3}$ entries $u = (x, y, z) \in T^*$ which get mapped to $i, j, k$, i.e. $f_1(x) = i, f_2(y) = j, f_3(z) = k$. Thus, as long as $7^s b_1 b_2 b_3 \gg d_1 d_2 d_3$, we should expect that the sampled matrix "covers" the original matrix, and so we get accurate answers. However, to analyze it formally, we need to take account of some potentially problematic dependencies between entries in the matrix. This was not an issue encountered in the original analysis [11], which only worked expectation-wise.

There are a number of other details to work out for the algorithms. For the real-valued case, we need certain weighting factors to account for the fact that some entries of the product $\bar{A} \boxtimes \bar{B}$ are over-represented in the the pseudo-multiplication. For the Boolean case, we need to have an additional randomization to handle the Boolean-to-GF(2) reduction. We provide the details next.

**Real-valued algorithm.** Having chosen the parameter $s$ and the random functions $f$, we will choose scaling matrices $G^A, G^B, G^C$ of dimensions $m_1 \times m_3, m_3 \times m_2, m_1 \times m_2$ respectively (their role will be discussed shortly). We then define

$$\bar{A}_{xz} = G^A_{xz} A_{f_1(x), f_3(z)}, \qquad \bar{B}_{zy} = G^B_{zy} B_{f_3(z) f_2(y)}$$

and, after computing the pseudo-product $\bar{C} = \bar{A} \boxtimes \bar{B}$ over $\mathbb{R}$, we estimate $C$ by

$$\tilde{C}_{ij} = \sum_{x \in f^{-1}(i), y \in f_2^{-1}(j)} G^C_{xy} \bar{C}_{xy}$$

Importantly, for values $x = (x', x'') \in \{0,1\}^s \times [b_1], z = (z', z'') \in \{0,1\}^s \times [b_2]$, the value of each term $G^A_{xz}$ should only depend $x', z'$, in particular, it should depend on the Hamming weights of $|x'|, |z'|, |x' \vee z'|$;. A similar criterion should hold for $G^B, G^C$ matrices. Thus, with

a slight abuse of notation, we write $G^A_{x',z'}$ and similarly $G^B_{z',y'}, G^C_{x',y'}$. Thus, the matrices $G$ really only have $O(s^3)$ degrees of freedom; we discuss later how to choose their values (both for asymptotic analysis and practical parameters).

**Boolean algorithm.** In this case, after drawing random functions $f$, we will also draw a uniformly random $m_3 \times m_2$ binary matrix $D$. We then form matrices $\bar{A}, \bar{B}$ by setting:

$$\bar{A}_{xz} = A_{f_1(x)f_3(z)}, \qquad \bar{B}_{zy} = B_{f_3(z)f_2(y)} D_{zy}$$

We then compute the pseudo-product $\bar{C} = \bar{A} \boxtimes \bar{B}$ over the finite field GF(2), with the given base case sizes, and produce an estimated matrix $\tilde{C}$ by $\tilde{C}_{ij} = \bigvee_{x \in f_1^{-1}(i), y \in f_2^{-1}(j)} \bar{C}_{xy}$.

**Analysis overview.** We will analyze the algorithms in three main steps. First, we derive some non-asymptotic bounds on the accuracy, in terms of parameters which can be computed explicitly as functions of $s$. Next, we use these to derive asymptotic bounds on the algorithm complexity. While illuminating as to the basic algorithm shape, these asymptotic bounds are not optimized and not of direct relevance for practical-scale computations. So we finish with some cases studies on very large problem parameters pushing the limits of practical computations, with concrete bounds.

Throughout, we define parameters

$$\psi_1 = d_1 + d_2 + d_3, \psi_2 = d_1 d_2 + d_2 d_3 + d_1 d_3, \psi_3 = d_1 d_2 d_3$$

Also, for each $\ell = 1, 2, 3$, we define $q_\ell = b_\ell / d_\ell$.

## 4 Analysis: the real-valued case

Consider some entry $i, j$; we can see that

$$\tilde{C}_{ij} = \sum_k \sum_{(x,y,z) \in T^*} G^A_{xz} G^B_{yz} G^C_{xy} [f_1(x) = i][f_2(y) = j][f_3(z) = k] A_{ik} B_{kj}$$

where we use Iverson notation for any boolean predicate here and throughout the paper, i.e. $[E] = 1$ if $E$ holds, otherwise $[E] = 0$. Our strategy to analyze $\tilde{C}_{ij}$ is to compute the mean and variance. We will show that $\mathbb{E}[\tilde{C}_{ij}] \propto C_{ij}$ (with known proportionality constants). Thus, by rescaling the matrix entries $G$ as needed, we can obtain an unbiased estimate of $C_{ij}$. Likewise, the variance of $\tilde{C}_{ij}$ will depend on $C_{ij}$ as well as its "second moment"

$$C^{(2)}_{ij} = \sum_k A^2_{ik} B^2_{kj}$$

To simplify the notation, let us write

$$G_{xyz} := G^A_{xz} G^B_{yz} G^C_{xy}$$

for a triple $(x, y, z) \in T$. We have the following main estimate:

▶ **Lemma 6.** *Define the sums $r_0, \dots, r_7$ by:*

$$r_0 = \sum_{(x,y,z)\in T_s} G_{xyz} \qquad r_1 = \sum_{x\in\{0,1\}^s} \Big( \sum_{y,z:(x,y,z)\in T_s} G_{xyz} \Big)^2$$

$$r_2 = \sum_{y\in\{0,1\}^s} \Big( \sum_{x,z:(x,y,z)\in T_s} G_{xyz} \Big)^2, \qquad r_3 = \sum_{z\in\{0,1\}^s} \Big( \sum_{x,y:(x,y,z)\in T_s} G_{xyz} \Big)^2$$

$$r_4 = \sum_{x,y\in\{0,1\}^s} \Big( \sum_{z:(x,y,z)\in T_s} G_{xyz} \Big)^2, \qquad r_5 = \sum_{x,z\in\{0,1\}^s} \Big( \sum_{y:(x,y,z)\in T_s} G_{xyz} \Big)^2$$

$$r_6 = \sum_{y,z\in\{0,1\}^s} \Big( \sum_{x:(x,y,z)\in T_s} G_{xyz} \Big)^2, \qquad r_7 = \sum_{(x,y,z)\in T_s} G_{xyz}^2$$

*Then, for any $i, j$, there holds $\mathbb{E}[\tilde{C}_{ij}] = q_1q_2q_3r_0C_{ij}$ and*

$$\mathbb{V}[\tilde{C}_{ij}] \le q_1q_2q_3^2(q_2r_1 + q_1r_2 + r_4)C_{ij}^2 + q_1q_2q_3(q_1q_2r_3 + q_2r_5 + q_1r_6 + r_7)C_{ij}^{(2)}$$

**Proof.** See full paper. ◀

For non-negative matrices, the statistic $\tilde{C}_{ij}$ can be used to estimate the value $C_{ij}$ up to some relative accuracy. We summarize this as follows:

▶ **Corollary 7.** *If matrices $A, B$ are non-negative, then*

$$\frac{\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2} \le \frac{(q_3q_2r_1 + q_3q_1r_2 + q_1q_2r_3) + (q_3r_4 + q_2r_5 + q_1r_6) + r_7}{q_1q_2q_3r_0^2}$$

**Proof.** We know $C_{ij} \ge 0$ and $C_{ij}^{(2)} \le C_{ij}^2$ for each entry $i, j$. ◀

## 5 Analysis: the Boolean case

In the Boolean case, it does not make sense to talk about how close the matrices entries $\tilde{C}_{ij}$ are to the true value $C_{ij}$. Instead, we will argue that, with high probability, we have $\tilde{C}_{ij} = C_{ij}$ exactly. It is clear that if $C_{ij} = 0$, then $\tilde{C}_{ij} = 0$ with probability one. Thus, consider some pair $i, j$ with $C_{ij} = 1$ and some witness value $k$ with $A_{ik} = B_{kj} = 1$. We want to find a triple of values $(x, y, z) \in T^*$ such that $f_1(x) = i, f_2(y) = j, f_3(z) = k$. If such a triple exists, then the randomization involved in the $D$ matrix should ensure that $\bar{C}_{xy} = 1$ with probability $1/2$, in which case $\tilde{C}_{ij} = 1$ as well.

To show the existence of such a triple $(x, y, z)$, we will use an important general probabilistic inequality of Janson [9]. We summarize it in the following form:

▶ **Theorem 8** ([9])**.** *Suppose that $X_1, \dots, X_N$ are independent Bernoulli variables, and suppose $\mathcal{E}$ is a collection of monomial events over ground set $[N]$, i.e. each event $E \in \mathcal{E}$ is a conjunction of the form $\bigwedge_{r\in R_E} X_r$ for a given subset $R_E \subseteq [n]$. For events $E, E' \in \mathcal{E}$, we write $E \sim E'$ if $R_E \cap R_{E'} \neq \emptyset$, and we write $\bar{E}$ for the complement of $E$ (i.e. that event $E$ does not hold.)*

*If we define*

$$\kappa = \sum_{E\in\mathcal{E}} \mathbb{E}\Big[\frac{[E]}{\sum_{E'\sim E}[E']}\Big]$$

*then we have*

$$\Pr(\bigwedge_{E\in\mathcal{E}} \bar{E}) \le e^{-\kappa}$$

We use it to get the following main result:

▶ **Theorem 9.** *Define values*

$$q'_1 = 1 - (1 - 1/d_1)^{b_1}, q'_2 = 1 - (1 - 0.5/d_2)^{b_2}, q'_3 = 1 - (1 - 1/d_3)^{b_3}$$

*Consider independent Bernoulli variables $I_{\ell,x}$ for $\ell = 1, 2, 3$ and $x \in \{0,1\}^s$, wherein each $I_{\ell,x}$ has expected value $q'_\ell$. For each tuple $u = (x, y, z) \in T_s$, define associated event $E_u$ to be the conjunction $I_{1,x} = I_{2,y} = I_{3,z} = 1$.*

*Then, for any pair $i, j$ with $C_{ij} = 1$, there holds* $\Pr(\tilde{C}_{ij} = 0) \leq \Pr(\bigwedge_{u \in T_s} \bar{E}_u)$.

**Proof.** We divide the analysis into two parts. first, we consider the random functions $f_\ell$, and then consider the random matrix $D$. Let us fix some arbitrary value $k$ with $A_{ik} = B_{kj} = 1$. Suppose that we chosen the functions $f_\ell$, and let $P$ denote the set of triples $(x, y, z) \in T^*$ with $f_1(x) = i, f_2(y) = j, f_3(z) = k$. From this, let $S$ denote the number of *distinct* values $y$ encountered, i.e. $S = |\{y : (x, y, z) \in P\}|$.

We claim that, having fixed the functions $f_\ell$, the probability that $C_{ij} = 0$ is at most $2^{-S}$. For, suppose that $P$ contains triples $(x_1, y_1, z_1), \ldots, (x_S, y_S, z_S)$ where $y_1, \ldots, y_S$ are distinct. In forming each entry $\bar{C}_{x_\ell y_\ell}$, we are adding in $D_{z_\ell y_\ell} A_{ik} B_{kj} = D_{z_\ell y_\ell}$ as well as potentially other entries $D_{z' y_\ell}$. These are all independent random bits, and hence their sum over GF(2) is equal to 1 with probability precisely $1/2$; in this case we would have $\bar{C}_{x_\ell y_\ell} = 1$ and hence $\tilde{C}_{ij} = 1$. Furthermore, since $y_1, \ldots, y_S$ are distinct, all such events are independent.

We also define independent Bernoulli random variables $I'_{1,x}, I'_{2,y}, I'_{3,z}$ to be the events that, respectively, $f_1(x) = i$, or $f_2(y) = j$, or $f_3(z) = k$; these have means $\frac{1}{d_1}, \frac{1}{d_2}, \frac{1}{d_3}$ respectively. Note that $P$, and hence $S$, is determined by these variables. Integrating over them, we have

$$\Pr(\tilde{C}_{ij} = 0) \leq \mathbb{E}[2^{-S}]$$

For each $x \in \{0,1\}^s$, we define events

$$I_{1,x} = \bigvee_{x'} I'_{1,(x,x')}, \qquad I_{2,y} = \bigvee_{y'} I'_{2,(y,y')} J_{(y,y')}, \qquad I_{3,z} = \bigvee_{z'} I'_{3,(z,z')},$$

where $x', y', z'$ range over $[b_1], [b_2], [b_3]$ respectively, and where $J_{(y,y')}$ are independent Bernoulli-1/2 random variables. Observe that $I_{1,x}, I_{2,y}, I_{3,z}$ are all independent Bernoulli random variables with means $q'_1, q'_2, q'_3$ respectively. Now consider the probability of the event $\bar{E}_u$. Suppose we reveal random variables $I'_{1,x}, I'_{2,y}, I'_{3,z}$. Conditional on these values, the only way for $\bigwedge_u \bar{E}_u$ to occur is that $J_y = 0$ holds for all tuples $u = (x, y, z) \in P$. This has probability precisely $2^{-S}$. Integrating over variables $I'$, we get:

$$\Pr(\bigwedge_u \bar{E}_u) = \mathbb{E}[2^{-S}]$$

Putting the inequalities together gives the claimed result

$$\Pr(\tilde{C}_{ij} = 0) \leq \mathbb{E}[2^{-S}] = \Pr(\bigwedge_u \bar{E}_u)$$ ◀

Observe that events $E_u$ of Theorem 9 are precisely the types of monomial events covered by Theorem 8; this will be used for asymptotic and concrete bounds later.

## 6 Asymptotic analysis

The asymptotic analysis of both the real-valued and Boolean use similar arguments. For this section, we let $b_1 = b_2 = b_3 = 1$, since the base cases should only affect analysis by constant factors.

We want to determine the number of triples $(x, y, z) \in T_s$ with $f_1(x) = i, f_2(y) = j, f_3(z) = k$ for values $i, j, k$; the relevant quantities ($r_0, \dots, r_7$ for the real-valued case and $\kappa$ for the Boolean case) are in effect counting these triples. Our basic strategy is to restrict the analysis to "typical" triples, specifically, let us define $H \subseteq T_s$ to be the set of triples $(x, y, z) \in T$ satisfying the following conditions:

1. $|x| \le 4s/7$ and $|y| \le 4s/7$ and $|z| \le 4s/7$
2. $|x \vee y| \le 6s/7$ and $|x \vee z| \le 6s/7$ and $|y \vee z| \le 6s/7$

▶ **Lemma 10.** *There holds* $|H| \ge \Omega(7^s)$.

**Proof.** There are precisely $7^s$ triples $(x, y, z)$ in $T_s$. Let us consider the probability space which is the uniform distribution on $T_s$; equivalently, for each coordinate $i$, the values $(x_i, y_i, z_i)$ are chosen uniformly at random among the 7 non-zero values. We need to show that the conditions on the densities of $x, y, z, x \vee y, x \vee z, y \vee z$ are satisfied with constant probability.

For each coordinate $i$, consider the 6-dimensional random vector $V_i = (x_i, y_i, z_i, x_i \vee y_i, x_i \vee z_i, y_i \vee z_i)$ and let $V = \sum_i V_i$. We need to show that $V - \beta s$ has non-positive entries where $\beta = (4/7, 4/7, 4/7, 6/7, 6/7, 6/7)$. For this, we use the multidimensional Central Limit Theorem. Since the variables $V_i$ are i.i.d. and each has mean $\beta$, the scaled value

$$\sqrt{s}(V/s - \beta)$$

approaches to a 6-dimensional normal distribution $N(0, \Sigma)$ with covariance matrix given by

$$\Sigma = \frac{1}{49}\begin{bmatrix} 12 & -2 & -2 & 4 & 4 & -3 \\ -2 & 12 & -2 & 4 & -3 & 4 \\ -2 & -2 & 12 & -3 & 4 & 4 \\ 4 & 4 & -3 & 6 & -1 & -1 \\ 4 & -3 & 4 & -1 & 6 & -1 \\ -3 & 4 & 4 & -1 & -1 & 6 \end{bmatrix}$$

as $s \to \infty$.

Note that $\Sigma$ is non-singular. Hence, for the corresponding distribution $N(0, \Sigma)$ there is positive probability that all six coordinates are negative. For large enough $s$, the probability that $V - \beta s$ has negative entries converges; in particular, it approaches to some constant value as $s \to \infty$. (The lemma holds vacuously when $s = O(1)$.) ◀

We will define two constants which show up repeatedly in the analysis:

$$\alpha_1 = 14 \cdot 2^{1/7} \cdot 3^{3/7} \approx 24.75$$

$$\alpha_2 = 7 \cdot 2^{6/7} \approx 12.68$$

▶ **Lemma 11.** *For* $i = 1, 2$, *there are* $O(\frac{\alpha_i^s}{\sqrt{s}})$ *pairs* $(x, y, z), (x', y', z') \in H$ *with* $[x = x'] + [y = y'] + [z = z'] \ge i$.

**Proof.** For $i = 1$, we will count the pairs $(x, y, z), (x', y', z') \in H$ with $x = x'$; the cases with the other two coordinates are completely symmetric. Let us suppose that, among the $s$ entries of $x$, there are $\ell$ coordinated equal to one; since $(x, y, z) \in H$ we must have $\ell \le 4s/7$.

For each coordinate $i$ with $x_i = x'_i = 1$, the values $y_i, y'_i, z_i, z'_i$ can be arbitrary (giving $16^\ell$ choices); for each coordinate $i$ with $x_i = x'_i = 0$, there are three non-zero choices for $(y_i, z_i)$ and three non-zero choices for $(y'_i, z'_i)$. Overall, there are $16^\ell 9^{s-\ell}$ choices for the vectors $y, z, y', z'$. Summing over $\ell$, we get that the total contribution is

$$\sum_{\ell=0}^{\lfloor 4s/7 \rfloor} \binom{s}{\ell} 16^\ell 9^{s-\ell}$$

Let $h(\ell) = \binom{s}{\ell} 16^\ell 9^{s-\ell}$ be the summand corresponding to $\ell$; we can observe that $h(\ell)/h(\ell-1) = \frac{16(s+1-\ell)}{9}$. In particular, for $\ell \le 4s/7$, this ratio is at least $4/3$. Hence, the entire sum $\sum_{\ell=0}^{4s/7} h(\ell)$ is within a constant factor of $h(4s/7)$. In turn, by Stirling's formula, we can calculate $h(4s/7) \le O(\alpha_1^s/\sqrt{s})$.

For $i = 2$, we count the pairs $(x, y, z), (x', y', z') \in H$ with $x = x', y = y'$; the cases with the other two coordinates are again completely symmetric. Let us suppose that, among the $s$ entries of $x, y$, there are $\ell$ coordinates with $x_i \vee y_i = 1$; since $(x, y, z) \in H$ we must have $\ell \le 6s/7$. For each such coordinate $i$, there are 4 choices for $z, z'$ and 3 choices for $x, y$; for each coordinate with $x_i \vee y_i = x'_i \vee y'_i = 0$, we must have $z_i = z'_i = 1$. Summing over $\ell$, the total contribution is

$$\sum_{\ell=0}^{\lfloor 6s/7 \rfloor} \binom{s}{\ell} 3^\ell 4^\ell$$

Let $h(\ell) = \binom{s}{\ell} 12^\ell$ be the summand corresponding to value $\ell$; note that $h(\ell)/h(\ell-1) = \frac{12(s+1-\ell)}{\ell}$. For $\ell \le 6s/7$, this ratio is at least 2, and hence the entire sum is within a constant factor of $h(6s/7)$. In turn, by Stirling's formula, we get $h(6s/7) \le O(\alpha^2 s/\sqrt{s})$. ◀

▶ **Proposition 12.** *Suppose we define matrices $G^A, G^B, G^C$ by*

$$G^A_{xz} = [|x| \le 4s/7][|z| \le 4s/7][|x \vee z| \le 6s/7]$$
$$G^B_{yz} = [|y| \le 4s/7][|z| \le 4s/7][|y \vee z| \le 6s/7]$$
$$G^C_{xy} = [|x| \le 4s/7][|y| \le 4s/7][|x \vee y| \le 6s/7]$$

*Then the real-valued algorithm satisfies*

$$\mathbb{E}[\tilde{C}_{ij}] = \nu C_{ij} 7^s/\psi_3 \qquad \text{for a known proportionality constant } \nu$$
$$\mathbb{V}[\tilde{C}_{ij}] = O\Big(q_1 q_2 q_3^2 (q_1+q_2)\alpha_1^s/\sqrt{s} + \alpha_2^s/\sqrt{s}\Big) C_{ij}^2 + O\Big(q_1 q_2 q_3 (q_1 q_2 \alpha_1^s/\sqrt{s} + (q_1+q_2)\alpha_2^s/\sqrt{s} + 7^s\Big) C_{ij}^{(2)}$$

**Proof.** With this definition of the matrices $G^A, G^B, G^C$, we have $G_{xyz} = [(x, y, z) \in H]$. Now let us compute the sums $r_0, \ldots, r_7$. For $r_0$, we have $r_0 = \sum_{(x,y,z)\in T_s} G_{x,y,z} = \sum_{(x,y,z)\in H} 1 = |H|$; by Lemma 10 this is $\Omega(7^s)$.

For $r_1$, we compute

$$r_1 = \sum_{x\in 0,1^s} \Big( \sum_{y,z:(x,y,z)\in T_s} G_{xyz} \Big)^2 = \sum_{x\in 0,1^s} \Big( \sum_{y,z:(x,y,z)\in H} 1 \Big)^2 = \sum_{\substack{(x,y,z)\in H,(x',y',z')\in H \\ x=x'}} 1$$

By Lemma 11, this is $O(\alpha_1^s/\sqrt{s})$. By completely analogous reasoning, we have $r_2, r_3 \le O(\alpha_1^s/\sqrt{s})$ and $r_4, r_5, r_6 \le O(\alpha_2^s/\sqrt{s})$. For $r_7$, we have

$$r_7 = \sum_{(x,y,z)\in T_s} G^2_{xyz} = \sum_{(x,y,z)\in H} 1 = |H| \le |T_s| \le 7^s$$

Now we directly apply Lemma 6. ◀

By collecting terms in Proposition 12, and noting that $q_i = 1/d_i$ for $i = 1, 2, 3$, we immediately get the following corollary:

▶ **Corollary 13.** *Suppose that* $C^{(2)}_{ij} \leq C^2_{ij}$, *and that* $(\alpha_1/7)^s/\sqrt{s} \leq O(\psi_3/\psi_1)$ *and* $(\alpha_2/7)^s/\sqrt{s} \leq O(\psi_3/\psi_2)$. *Then*

$$\frac{\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2} \leq O(\psi_3/7^s)$$

▶ **Theorem 14.** *Suppose that matrices* $A, B$ *are non-negative. Let* $\epsilon \in (0, 1)$, *and define parameters* $\gamma = \psi_3/\epsilon^2$ *and* $\beta_1 = \psi_3/\psi_1$ *and* $\beta_2 = \psi_3/\psi_2$. *With appropriate choice of parameters, we can obtain an estimated matrix* $\tilde{C}$, *such that, for any entry* $i, j$, *there holds*

$$\Pr(\tilde{C}_{ij} \in [(1-\epsilon)C_{ij}, (1+\epsilon)C_{ij}]) \geq 3/4,$$

*with overall runtime*

$$O\Big(\gamma^{\frac{\log 6}{\log 7}} + \gamma\big((\beta_1\sqrt{\log\beta_1})^{\frac{\log(6/7)}{\log(\alpha_1/7)}} + (\beta_2\sqrt{\log\beta_2})^{\frac{\log(6/7)}{\log(\alpha_2/7)}}\big)\Big)$$

**Proof.** Our strategy will be to take multiple independent executions and average them so that the resulting sample means have relative variance $\epsilon^2/4$. By Chebyshev's inequality, the resulting sample means are then within $(1 \pm \epsilon)C_{ij}$ with probability at least $3/4$. To that end, let us set

$$s = \Big\lfloor \min\{\log_7 \gamma, \log_{\alpha_1/7}(\beta_1\sqrt{\log\beta_1}), \log_{\alpha_2/7}(\beta_2\sqrt{\log\beta_2})\}\Big\rfloor$$

It can be shown that $(\alpha_1/7)^s/\sqrt{s} \leq O(\psi_3/\psi_1)$ and $(\alpha_2/7)^s/\sqrt{s} \leq O(\beta_2)$. Now, by collecting terms in Proposition 12, and noting that $q_i = 1/d_i$ for $i = 1, 2, 3$, and using the bounds $(\alpha_1/7)^s/\sqrt{s} \leq O(\psi_3/\psi_1)$ and $(\alpha_2/7)^s/\sqrt{s}$ as well as the observation that $C^{(2)}_{ij} \leq C_{ij}$, we get the estimate:

$$\frac{\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2} \leq O(\psi_3/7^s)$$

Thus, if we repeat the process for $\Omega(1 + \epsilon^{-2}/(7^s/\psi_3))$ iterations and take the mean of all observations, the overall relative variance of the resulting sample mean is reduced to $\epsilon^2/4$, and hence satisfies the required bounds. (We can scale by a known proportionality constant to get an unbiased estimator). Overall, the runtime is then

$$O(6^s(1 + \epsilon^{-2}/(7^s/\psi_3)))$$

which after some simplifications, gives the claimed runtime bounds. ◀

▶ **Corollary 15.** *Suppose matrices* $A, B$ *are non-negative. With appropriate choice of parameters, the real-valued algorithm can compute* $C$ *with relative error* $1 \pm \epsilon$ *with probability* $3/4$ *in runtime*

$$O\Big((\psi_3/\epsilon^2)^{\frac{\log 6}{\log 7}} + \frac{\psi_1^{0.122}\psi_3^{0.878} + \psi_2^{0.259}\psi_3^{0.741}}{\epsilon^2}\Big)$$

*In particular, if the matrix is square of dimension* $n$, *then the runtime is* $O(n^{2.77}/\epsilon^2)$.

As usual, we can boost the success probability to any desired value $1-\delta$ by using median amplification, with a further $O(\log(1/\delta))$ increase in runtime. Despite the (relatively) crisp formula, Corollary 15 is not directly relevant to practical parameter sizes. Many of the hidden constants are large, and the bounds are all somewhat crude. Since our main goal is to obtain a *practical* algorithm we need to consider more finely how the algorithm scales for *small* values of $n$ and $s$; we consider this in the next section.

The analysis for the Boolean case is similar to the real-valued case.

▶ **Theorem 16.** *Let $\delta \in (0,1)$, and define parameters $\gamma = \psi_3 \log(1/\delta)$ and $\beta_1 = \psi_3/\psi_1$ and $\beta_2 = \psi_3/\psi_2$. With appropriate choice of parameters, we can obtain an estimated matrix $\tilde{C}$, such that, for any entry $i, j$, there holds*

$$\Pr(\tilde{C}_{ij} = C_{ij}) \geq 1 - \delta,$$

*with overall runtime*

$$O\Big(\gamma^{\frac{\log 6}{\log 7}} + \gamma\big((\beta_1\sqrt{\log\beta_1})^{\frac{\log(6/7)}{\log(\alpha_1/7)}} + (\beta_2\sqrt{\log\beta_2})^{\frac{\log(6/7)}{\log(\alpha_2/7)}}\big)\Big)$$

**Proof.** The proof is completely analogous to Theorem 14 and is omitted. ◀

▶ **Corollary 17.** *With appropriate choice of parameters, we can compute the overall matrix $C$ correctly with probability at least $1 - 1/\operatorname{poly}(\psi_3)$ using runtime*

$$O\Big((\psi_3 \log \psi_3)^{\frac{\log 6}{\log 7}} + \psi_1^{0.122}\psi_3^{0.878} + \psi_2^{0.259}\psi_3^{0.741}\Big)$$

We mention one important difference in our analysis of the real-valued and Boolean cases. For both algorithms, the analysis depends on the set $H \subseteq T_s$. For the Boolean algorithm, this is only used for purposes of analysis; the actual algorithm does not need to use $H$. For the real-valued algorithm, we need $H$ to define the matrices $G^A, G^B, G^C$ (which are part of the algorithm itself)

## 7 Concrete complexity estimates

Since our goal is to get a new practical algorithm, the asymptotic estimates are of limited value on their own. In this section, we consider costing the matrix multiplication algorithms for problem sizes which are at the upper limits of a practical problem size. In particular, we want to compare our algorithm to Strassen's algorithm and the KK algorithm. There are two main issues to consider. First, we need a realistic estimate of the runtime; second, we need a realistic estimate of the algorithm accuracy and/or success probability. The asymptotic estimates are much too crude for our purposes here.

Let us first consider the runtime. The work for the algorithm will involve $6^s$ many iterations of the recursive partitioning as given in Theorem 3. The cost of this step will have a quadratic component (adding and rearranging the relevant submatrices) as well as a nearly-cubic component (coming from the recursive subproblems). Assuming that the base case parameters are chosen to be sufficiently large, the first of these should be negligible; overall, we can estimate the work as $6^s W_b$ where $W_b$ is the base-case cost.

Let us compare this runtime with other matrix-multiplication algorithms. To avoid issues with padding, let us assume for convenience that $d_1 = d_2 = d_3$ and $d$ is a power of two. In this case, both Strassen's algorithm and the KK algorithm would run $s$ many recursive iterations, and then again pass to a base case. Assuming again that the base case is sufficiently large to hide the quadratic parts of the algorithm, these would have costs of $7^s W_b$ and $6^s W_b$ respectively.

Estimating $W_b$ itself is much trickier. As a starting point, we might estimate $W_b \approx b_1 b_2 b_3$ in terms of arithmetic operations. But there are many additional factors to consider: a CPU with SIMD registers of width $w$ may be able to perform $w$ operations in parallel, which is especially powerful for GF(2) arithmetic. Furthermore, for GF(2), there are other optimizations such as Four-Russians. Fortunately, in order to compare our algorithm with Strassen or KK, the precise value of $W_b$ does not matter; it is common to all the algorithms.

To simplify further, let us assume that $b_1 = b_2 = b_3$, and let $b$ denote this common value, which we regard as a fixed constant. In this case, for Strassen and KK, the parameter $s$ must be set to a fixed value $s = s_0 = \log_2(n/b)$. Summarizing, the runtimes of our algorithm, Strassen's algorithms, and the KK algorithm, are proportional to values $6^s, 7^{s_0}, 6^{s_0}$, where $s_0 = \log_2(n/b)$, respectively.

At this point, let us bring up another important issue with costing the algorithms. As $s$ increases, the memory used by the algorithm also increases. Storing the original matrices requires roughly $n^2$ memory, and both Strassen and KK algorithms use essentially this same amount of memory as well. Our algorithm, by contrast, may require $6^s$ memory to store the expanded $\bar{A}, \bar{B}$; if $s$ is large, this may significantly increase the memory cost. Note that, for very large scale problems, the computational costs of the algorithm (i.e. total number of arithmetic operations) are dominatedc by the communication costs (i.e. moving memory across multiple nodes of a computer cluster.) In order to keep this section relatively contained, we will not investigate further the memory costs.

Next, we need to analyze the algorithm accuracy. Let us first consider the Boolean algorithm, where the comparison between our algorithm and the other candidate algorithms is more straightforward. Assuming we are using the reduction from Boolean matrix multiplication to GF(2) matrix multiplication, all three algorithms are randomized. Let us denote by $f$ the maximum failure probability for any entry $C_{ij}$, if we use the matrix multiplication algorithm directly. Assuming we want to boost the algorithm to a much lower failure probability $\delta$, we need to repeat for $\lceil \log(1/\delta)/\log(1/f) \rceil$ iterations. Indeed, if we want to compute the entire matrix $C$ correctly, the value $\delta$ may need to be quite small (on the order of $1/n^2$). In order to put the algorithms on an even footing, let us therefore define a cost parameter

$$M = \frac{\text{Runtime}}{\log(1/f)} \tag{1}$$

The value $f$ may depend on the number of witness $k$ with $A_{ik}B_{kj} = 1$. The extremal case for all the algorithms appears to be the case where there is only one witness. In this case, the KK algorithm has failure probability $f = 1 - (7/8)^s/2$ and Strassen's algorithm has failure probability $1/2$. We thus have

$$M_{\text{Strassen}} = 7^s/\log 2, \qquad M_{\text{KK}} = \frac{6^s}{-\log(1-(7/8)^s/2)} \sim 2 \cdot (48/7)^s \approx 2 \cdot 6.857^s$$

To compare, we will compute the value $M$ for our algorithm in the next section.

Next, let us consider a scenario for real-valued matrix multiplication. To make things concrete, let us suppose that $A, B$ are non-negative matrices and we want to estimate the value $C_{ij}$ up to $(1 \pm \epsilon)$ for some constant value $\epsilon = 1/2$ and with success probability $3/4$. In order to do so, we need to run the matrix multiplication algorithm for

$$L = \frac{4\mathbb{V}[\tilde{C}_{ij}]}{\epsilon^2 \mathbb{E}[\tilde{C}_{ij}]^2} = \frac{16\mathbb{V}[\tilde{C}_{ij}]}{\mathbb{E}[\tilde{C}_{ij}]^2}$$

trials. The runtime for our algorithm will thus be

$$6^s \cdot 16\mathbb{V}[\tilde{C}_{ij}]/\mathbb{E}[\tilde{C}_{ij}]^2$$

This comparison is among the favorable possible for our algorithm. Strassen's algorithm has error probability zero and has perfect accuracy.

Although [11] did not discuss it, it is also possible to use KK for real-valued matrix multiplication (where we assume a random permutation of the rows and columns of $A, B$). We can estimate its runtime as $6^{s_0} \cdot 16(8/7)^{s_0}$, while Strassen's algorithm will take time $7^{s_0}$.

## 7.1 Case study: non-negative matrix multiplication

Let us consider a sample problem instance with $d_1 = d_2 = d_3 = 2^{25}$; as a reasonable choice for base case, let us take $b_1 = b_2 = b_3 = 2^7$. Note that storing the matrices already requires $2^{51}$ words of memory, which is at the upper limit of practical. Beyond this size, the problem would likely require a highly distributed memory system, and the runtime would be mostly determined by scheduling communications (as opposed to counting arithmetic operations).

With these parameters, Strassen and KK will set $s_0 = 18$, and they have runtimes of respectively

$$2^{50.5}, 2^{54.0}$$

In order to compare our algorithm to these two algorithms, to need to choose parameter $s$ and we also need to determine the matrices $G^A, G^B, G^C$. We do not know how to select the matrices $G$ in the optimal way, and the choice in Proposition 12 would be suboptimal by large constant factors. For our purposes, we used a local search method to select $G$, in order to minimize relative variance. (We note that, if instead we simply fixed all entries of $G^A, G^B, G^C$ to be equal to one, the variance would only increase by roughly 10% – 20%)

Figure 1, following, shows the relative variance and runtime for various choices of parameter $s$, and for our chosen accuracy parameter $\varepsilon = 1/2$ and success probability 3/4.

| $s$ | $\mathbb{V}[\tilde{C}_{ij}]/\mathbb{E}[\tilde{C}_{ij}]^2$ | Runtime |
|---|---|---|
| 10 | 25.9 | 55.8 |
| 11 | 23.1 | 55.6 |
| 12 | 20.3 | 55.4 |
| 13 | 17.6 | 55.2 |
| 14 | 14.8 | 55.0 |
| 15 | 12.1 | 54.8 |
| 16 | 9.4 | 54.8 |
| 17 | 6.9 | 54.8 |
| 18 | 4.5 | 55.0 |
| 19 | 2.4 | 55.6 |
| 20 | 0.7 | 56.4 |
| 21 | −0.7 | 57.6 |

**Figure 1** Runtime of our algorithm for the sample problem. To handle the wide dynamic range, all figures are given in log base two; e.g., for $s = 10$, relative variance is $2^{25.9}$ and runtime is $2^{55.8}$.

The optimal value appears to be roughly $s \approx 15$, with runtime $2^{54.8}$. This is slightly worse than KK algorithm, and much worse than Strassen's algorithm. We emphasize that this comparison is still unfair to Strassen's algorithm, which provides precise and deterministic calculations of $C_{ij}$. Also, the base case choice here is still probably smaller than optimal, again making the comparisons unfairly favorable to the randomized algorithms.

Thus, despite the asymptotic advantages, it will probably never be profitable to our algorithm, or the KK algorithm, for real-valued matrix multiplication.

## 7.2 Estimating performance for Boolean matrix multiplication

We now turn to estimating the failure probability of our algorithm for the Boolean setting. Unlike the KK algorithm, the random process involved is much more complex and we cannot obtain a simple closed-form expression. Our starting point is to consider the random process of Theorem 9, and to try to estimate the probability $p$ of the event $\bigwedge_{u \in T_s} \bar{E}_u$; note that $f \leq p$.

Let us consider a sample problem with $d_1 = d_2 = d_3 = 2^{25}$; following heuristics of [1], we might take the base case size as $b_1 = b_2 = b_3 = 2^{10}$. With these parameters, we can calculate

$$M_{\text{Strassen}} = 2^{42.64}, M_{\text{KK}} = 2^{42.61}$$

with both algorithms using $s_0 = 15$. (Recall the definition of $M$ from Eq. (1).) To compare these with our algorithm, we use both empirical simulations (with $10^6$ trials) on the hand, and Theorem 8 combined with Theorem 9 on the other hand, to obtain estimates of failure probability. Specifically, we write $\tilde{\kappa}$ for the estimate coming from Theorem 8 and we write $\hat{\kappa} = \log(1/\hat{p})$ for the Monte Carlo estimate.

By way of comparison, we can also calculate $\mu = 7^s q_1' q_2' q_3'$ to be the expected number of events $E_u$ which hold. In an ideal setting, if the events $E_u$ were completely independent, the failure probability would be $e^{-\mu}$, and we would have $\kappa = \mu$. See Figure 2.

| $s$ | $\mu$ | $\tilde{\kappa}$ | $\hat{\kappa}$ | $M$ (upper bound) | $M$ (empirical) |
|---|---|---|---|---|---|
| 7 | −26.3 | −26.4 | | 44.5 | |
| 8 | −23.5 | −23.6 | | 44.2 | |
| 9 | −20.7 | −20.8 | −19.9 | 44.0 | 43.2 |
| 10 | −17.9 | −18.0 | −18.9 | 43.8 | 44.8 |
| 11 | −15.1 | −15.2 | −15.2 | 43.6 | 43.5 |
| 12 | −12.3 | −12.5 | −12.5 | 43.5 | 43.5 |
| 13 | −9.5 | −9.8 | −9.7 | 43.4 | 43.3 |
| 14 | −6.7 | −7.3 | −7.1 | 43.5 | 43.2 |
| 15 | −3.9 | −5.0 | −4.5 | 43.7 | 43.3 |
| 16 | −1.1 | −2.9 | −2.1 | 44.3 | 43.5 |
| 17 | 1.7 | −1.2 | −0.1 | 45.2 | 44.0 |
| 18 | 4.5 | 0.2 | 1.6 | 46.4 | 44.9 |
| 19 | 7.3 | 14 | 3.0 | 47.7 | 46.1 |
| 20 | 10.1 | 2.5 | | 49.2 | |
| 21 | 13.0 | 3.5 | | 50.8 | |

**Figure 2** Possible failure parameters for the algorithm. Figures are given in log base two. Some empirical estimates are left blank because statistically significant estimates could not be obtained.

The optimal value $s$ appears to be roughly $s = 14$, and for this value we indeed have $\tilde{\kappa} \approx \hat{\kappa}$. Thus, a good heuristic to select $s$ would be to use the calculated value $\tilde{\kappa}$ and choose $s$ to minimize the resulting value $M$. This is much faster than empirical estimation, and it is also safer in that the resulting failure probability is a provable upper bound on the true failure probability.

## 8 Potential improvements

In the algorithms as we describe, the hash functions $f_\ell$ are chosen uniformly at random. The number of preimages of any $i \in [d_\ell]$ is thus a Binomial random variable with mean

$$\mu = \frac{2^s b_\ell}{d_\ell} = 2^s q_\ell$$

There are likely better ways to choose the random hash functions. For example, one could use a dependent rounding scheme to ensure that each $i \in [d_\ell]$ has precisely $\lfloor \mu \rfloor$ or $\lceil \mu \rceil$ preimages. One could even go further, aiming to ensure that the set of preimages $x \in f_\ell^{-1}(i)$ is "balanced" in terms of the Hamming weight of its elements.

These types of modifications would likely reduce the variance of the statistical estimates we develop. For practical parameter sizes, they could make a critical difference in the performance of the algorithm. However, these modifications are also much harder to analyze precisely. We leave as an open question the benefit of such improvements and whether they can make this algorithm fully practical.

## References


1 Martin Albrecht, Gregory Bard, and William Hart. Algorithm 898: Efficient multiplication of dense matrices over $GF(2)$. *ACM Transactions on Mathematical Software (TOMS)*, 37(1):1–14, 2010.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

3 Vladimir L'vovich Arlazarov, Yefim A Dinitz, MA Kronrod, and IgorAleksandrovich Faradzhev. On economical construction of the transitive closure of an oriented graph. In *Doklady Akademii Nauk*, volume 194, pages 487–488. Russian Academy of Sciences, 1970.

4 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 193–204, 2012.

5 Austin R Benson and Grey Ballard. A framework for practical parallel fast matrix multiplication. *ACM SIGPLAN Notices*, 50(8):42–53, 2015.

6 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

7 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

8 Jianyu Huang, Tyler M Smith, Greg M Henry, and Robert A Van De Geijn. Strassen's algorithm reloaded. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 690–701. IEEE, 2016.

9 Svante Janson, Tomasz Luczak, and Andrzej Rucinski. An exponential bound for the probability of nonexistence of a specified subgraph in a random graph. In *Random graphs*, volume 87, pages 73–87, 1990.

10 Matti Karppa. Techniques for similarity search and Boolean matrix multiplication. PhD thesis, 2020.

11 Matti Karppa and Petteri Kaski. Probabilistic tensors and opportunistic Boolean matrix multiplication. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–515. SIAM, 2019.

**12** François Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 514–523. IEEE, 2012.

**13** Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. *Information and Computation*, 261:240–247, 2018.

# Counting and Sampling Labeled Chordal Graphs in Polynomial Time


**Úrsula Hébert-Johnson**
University of California, Santa Barbara, CA, USA

**Daniel Lokshtanov**
University of California, Santa Barbara, CA, USA

**Eric Vigoda**
University of California, Santa Barbara, CA, USA



## Abstract

We present the first polynomial-time algorithm to exactly compute the number of labeled chordal graphs on $n$ vertices. Our algorithm solves a more general problem: given $n$ and $\omega$ as input, it computes the number of $\omega$-colorable labeled chordal graphs on $n$ vertices, using $O(n^7)$ arithmetic operations. A standard sampling-to-counting reduction then yields a polynomial-time exact sampler that generates an $\omega$-colorable labeled chordal graph on $n$ vertices uniformly at random. Our counting algorithm improves upon the previous best result by Wormald (1985), which computes the number of labeled chordal graphs on $n$ vertices in time exponential in $n$.

An implementation of the polynomial-time counting algorithm gives the number of labeled chordal graphs on up to 30 vertices in less than three minutes on a standard desktop computer. Previously, the number of labeled chordal graphs was only known for graphs on up to 15 vertices.



**2012 ACM Subject Classification** Theory of computation → Generating random combinatorial structures; Theory of computation → Graph algorithms analysis

**Keywords and phrases** Counting algorithms, graph sampling, chordal graphs

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.58

**Related Version** *Full Version*: `https://arxiv.org/abs/2308.09703`

**Supplementary Material** *Software (Source Code)*: `https://github.com/uhebertj/chordal` archived at `swh:1:dir:b81faedf15e1e8ebd4b68f421d2547ed39e1cd61`

**Funding** *Úrsula Hébert-Johnson*: Supported by NSF grant CCF-2008838.
*Daniel Lokshtanov*: Supported by NSF grant CCF-2008838.
*Eric Vigoda*: Supported by NSF grant CCF-2147094.


## 1 Introduction

Generating random graphs from a prescribed graph family is a fundamental task for running simulations and testing conjectures. Although generating a random labeled graph on $n$ vertices is easy (just flip an unbiased coin for each potential edge), the first polynomial-time algorithm for generating an *unlabeled* graph uniformly at random was only given in 1987, by Wormald [36]. The algorithm of Wormald runs in polynomial time in expectation, and to the best of our knowledge, the existence of a worst-case polynomial-time sampler of random unlabeled graphs remains open.

Naturally, when we wish to sample from a specified graph family, there are many interesting families of graphs for which this problem is nontrivial, even when the graphs are labeled. For the class of labeled trees, a sampling algorithm using Prüfer sequences [27] was discovered in 1918. More recently, a fast (exact) uniform sampler was presented by Gao and Wormald for $d$-regular graphs with $d = o(\sqrt{n})$ in 2017 [11], and then for power-law graphs in 2018 [12]. A





more general problem is the following: given an arbitrary degree sequence, generate a random graph with those specified degrees – this has been resolved for bipartite graphs [21, 3] as well as for general graphs when the maximum degree is not too large [16, 1]. See Greenhill [15] for a survey of random generation of graphs with degree constraints. For planar graphs, Bodirsky, Gröpl, and Kang presented a polynomial-time algorithm [6], which uses dynamic programming to exactly compute the number of labeled planar graphs on $n$ vertices and generate a planar graph uniformly at random in time $\widetilde{O}(n^7)$. This was improved to $O(n^2)$ expected time by Fusy [10], using a Boltzmann sampler.

Our results fall naturally within this line of work. We consider the problem of generating a labeled *chordal* graph on $n$ vertices uniformly at random. A graph is chordal if it has no induced cycles of length at least 4. Despite being one of the most fundamental and well-studied graph classes, prior to our work, the fastest uniform sampling algorithm for labeled chordal graphs was the exponential-time algorithm of Wormald from 1985 [35]. (To be precise, optimizing the running time of an algorithm for counting chordal graphs was not the main focus of Wormald; rather, the main goal of the paper was to determine the asymptotic number of chordal graphs with given connectivity, and the exponential-time algorithm is a corollary of these results.) Since then, various algorithmic approaches have been proposed for generating chordal graphs (e.g., [23, 32, 33, 8, 24]), but these algorithms do not come with any formal guarantees about their output distribution. In particular, [32] specifically asks for the existence of a polynomial-time algorithm to sample chordal graphs uniformly at random as an open problem. In a recent abstract, Sun and Bezáková [33] proposed a Markov chain for sampling chordal graphs, but this Markov chain comes with few mixing time guarantees.

We obtain the first polynomial (in $n$) time algorithm to exactly count the number of labeled chordal graphs on $n$ vertices, as well as the first polynomial-time uniform sampling algorithm for the class of labeled chordal graphs. Our algorithm also easily extends to counting and sampling $\omega$-*colorable* labeled chordal graphs. A graph $G$ is $\omega$-colorable if there exists a function $c\colon V(G) \to \{1, \dots, \omega\}$ such that every edge $uv \in E(G)$ satisfies $c(u) \neq c(v)$.

▶ **Theorem 1.** *There is a deterministic algorithm that given positive integers $n$ and $\omega \leq n$, computes the number of $\omega$-colorable labeled chordal graphs on $n$ vertices using $O(n^7)$ arithmetic operations. Moreover, there is a randomized algorithm that generates a graph uniformly at random from the set of all $\omega$-colorable labeled chordal graphs on $n$ vertices using $O(n^7)$ arithmetic operations.*

By the known equivalence between chromatic number, maximum clique size, and treewidth of chordal graphs [5], Theorem 1 can be reinterpreted as counting and sampling labeled chordal graphs of clique size at most $\omega$, or treewidth at most $\omega - 1$. The running time bound of Theorem 1 is stated in terms of the number of arithmetic operations. Since there are at most $2^{n^2}$ labeled graphs on $n$ vertices, the arithmetic operations need to deal with $n^2$-bit integers. Therefore, using the $O(n \log n)$-time algorithm for multiplying two $n$-bit integers [19] yields an $O(n^9 \log n)$-time upper bound for our algorithm in the RAM model.

A straightforward implementation of our counting algorithm gives the number of labeled chordal graphs on up to $n = 30$ vertices in less than three minutes on a standard desktop computer. Previously, the number of labeled chordal graphs was only known for graphs on up to 15 vertices [25]. In addition, we use our implementation to compute the number of $\omega$-colorable labeled chordal graphs for $n \leq 12$ and $\omega \leq 12$. We chose to stop at $n = 12$ to keep the table at a reasonable size, not because of the computation time. We present the computational results in Section 4.

## 1.1 A Brief Survey on Chordal Graphs

The literature on chordal graphs is so vast that it would be impossible to fully do it justice. Discussions of chordal graphs in the literature go as far back as 1958 [18]. What follows is a summary of some of the most notable problems and milestones.

Many NP-hard optimization problems (such as *coloring* [13] and *maximum independent set* [9]), as well as #P-hard counting problems (such as *independent sets* [26, 4]), and many others [29], are polynomial-time solvable on chordal graphs. Chordal graphs have a wide variety of applications, including phylogeny in evolutionary biology [17, 28] and Bayesian networks in machine learning [34]. When doing Gaussian elimination on a symmetric matrix, the set of matrix entries that are nonzero for at least one time-point of the elimination process corresponds to the edge set of a chordal graph. Thus the problem of finding an ordering in which to do Gaussian elimination that minimizes the number of nonzero matrix entries can be reduced to finding a chordal supergraph of a given graph with the minimum number of edges [30]. This problem, known as *minimum fill-in*, was shown to be NP-complete by Yannakakis [37]. Chordal graphs have a central place in graph theory [7], both through their connection to treewidth [20] and through their connection to perfect graphs [14]. From an algorithms perspective, chordal graphs can be recognized in linear time [31].

An interesting and relevant result by Bender et al. [2] is that a random $n$-vertex labeled chordal graph is a split graph with probability $1 - o(1)$, i.e., the fraction of labeled chordal graphs that are not split is $o(1)$. This yields a simple *approximately uniform* sampler for labeled chordal graphs: simply sampling a random labeled split graph leads to an output distribution with total variation distance $o(1)$ from the uniform distribution on labeled chordal graphs. This simple sampling algorithm is unsatisfactory because it can never output a non-split chordal graph. Nevertheless, this result suggests two things: The first is that it might be possible to find a simple and efficient uniform random sampler for labeled chordal graphs. The second is that the type of chordal graphs that one usually envisions when thinking of a chordal graph (namely those with relatively small treewidth) are different from those most likely to be generated by a uniform random sampler (namely split graphs). Therefore, to generate the type of chordal graphs that one usually envisions, one should be sampling not from the set of all chordal graphs but rather from a subset, e.g., the set of all $\omega$-colorable chordal graphs. Fortunately, Theorem 1 provides this functionality.

## 1.2 Methods

Our exact counting algorithm is based on dynamic programming. While clique trees and tree decompositions are never explicitly mentioned in the description of the algorithm, the intuition behind the algorithm is based on these notions. Essentially, we generate a rooted clique tree where the dynamic-programming table encodes certain properties of the graph, including how it relates to the root node of the clique tree. A *clique tree* of a graph $G$ is a tree $T$ together with a function $f$ that maps each vertex of $G$ to a connected vertex subset of $T$, such that for every pair $u$,$v$ of vertices in $G$, $uv$ is an edge in $G$ if and only if $f(u) \cap f(v) \neq \emptyset$. It is well known that a graph $G$ has a clique tree if and only if it is chordal [5].

The main difficulty with this approach is that different chordal graphs have different numbers of clique trees, so if we count the total number of clique trees, this will not give us an accurate count of the number of $n$-vertex chordal graphs. Therefore, the key idea behind our algorithm is to assign to each labeled chordal graph a unique "canonical" clique tree and to only count these canonical clique trees. The information stored in the dynamic-programming table is sufficient to ensure that every clique tree that we do count is the canonical tree of

some chordal graph, and that the canonical clique tree of every chordal graph is counted. As it turns out, the best way to phrase our algorithm is not in terms of clique trees at all, but rather in terms of an (essentially) equivalent notion that we call an "evaporation sequence." An evaporation sequence is closely related to the standard notion of a perfect elimination ordering (PEO) of a chordal graph. A *simplicial* vertex is a vertex whose neighborhood is a clique, and a PEO is an ordering of the vertices such that each is simplicial in the current induced subgraph, if we delete the vertices in that order (see Section 3.1 for more details). An evaporation sequence is a type of "canonical" PEO: at each step, we remove *all* simplicial vertices from the current subgraph, rather than making an arbitrary choice of a single simplicial vertex. We say that all of the simplicial vertices evaporate at time 1. Next, all of the vertices that become simplicial once the first set of simplicial vertices has been removed are said to evaporate at time 2, and so on. It is easy to see that every labeled chordal graph has a unique evaporation sequence, and that this sequence does not depend on the labeling of the vertices.

Therefore, the number of chordal graphs on $n$ vertices is the sum over all evaporation sequences of the number of labeled chordal graphs with that evaporation sequence. While different evaporation sequences correspond to different numbers of chordal graphs, because this number is independent of the labels, we can simply guess the *number* $x$ of vertices that evaporate at any given time, and then without loss of generality assign the labels $1, \ldots, x$ to those vertices.

In our dynamic-programming algorithm, the recursive subproblems deal with counting *rooted* clique trees. In this context, the root of a clique tree is the set of vertices that evaporate last. Just as we would like to "force" a set of nodes to be in the root of the tree, we will sometimes want to force a set of nodes to evaporate last. This is done using what we call an *exception set*, i.e., a set of vertices that do not evaporate even if they are simplicial.

The random sampling algorithm in Theorem 1 follows from our counting algorithm using the standard sampling-to-counting reduction of [22].

## 1.3 Overview of the Paper

Our dynamic-programming algorithm, including the associated recurrences, is presented in Section 3. For a glimpse of the proof of correctness, one can find the proof of the first recurrence (the reduction to counting *connected* chordal graphs) in Section 3.4. The complete proof of correctness, as well as the details of how to obtain the random sampling algorithm using the counting algorithm, can be found in the full version of the paper.

# 2 Preliminaries

Our algorithm counts vertex-labeled chordal graphs. For simplicity of notation, we assume the vertex set of each graph is a subset of $\mathbb{N} = \{1, 2, 3, \ldots\}$, which allows the labels to also serve as the names of the vertices. For example, we will speak of the vertex $5 \in V(G)$ rather than a vertex $v$ with label 5.

▶ **Definition 2.** *A* labeled graph *is a pair $G = (V, E)$, where the vertex set $V$ is a finite subset of $\mathbb{N}$ and the edge set $E$ is a set of two-element subsets of $V$.*

Henceforth, we implicitly assume *all graphs that we consider are labeled graphs.* For nonnegative integers $n$, we use the notation $[n] \coloneqq \{1, 2, \ldots, n\}$. Intervals of integers will often appear in our algorithm as the vertex set of a graph or subgraph, so we also define

$$[a, b] \coloneqq \{a, a + 1, \ldots, b\}$$

for nonnegative integers $a, b$. If $b < a$, then $[a, b] = \emptyset$.

▶ **Definition 3.** *Let $A = \{a_1, \ldots, a_r\}$ and $B = \{b_1, \ldots, b_r\}$ be finite subsets of $\mathbb{N}$ such that $|A| = |B|$, where the elements $a_i$ and $b_i$ are listed in increasing order. We define $\phi(A, B)\colon A \to B$ as the bijection that maps $a_i$ to $b_i$ for all $i \in [r]$.*

▶ **Definition 4.** *Let $G_1$, $G_2$ be two graphs, and suppose $C \coloneqq V(G_1) \cap V(G_2)$ is a clique in both $G_1$ and $G_2$. When we say we* glue $G_1$ and $G_2$ together at $C$ *to obtain $G$, this indicates that $G$ is the union of $G_1$ and $G_2$: the vertex set is $V(G) = V(G_1) \cup V(G_2)$, and the edge set is $E(G) = E(G_1) \cup E(G_2)$.*

For a graph $G$ and a vertex subset $S \subseteq V(G)$, $G[S]$ denotes the induced subgraph on the vertices of $S$. For a vertex $v \in V(G)$, we denote the neighborhood of $v$ in $G$ by $N_G(v)$, or by $N(v)$ if the graph is clear from the context. For $S \subseteq V(G)$, the open neighborhood of $S$ is denoted by

$$N_G(S) \coloneqq \{v \in V(G) \setminus S : uv \in E(G) \text{ for some } u \in S\}$$

and the closed neighborhood of $S$ is denoted by $N_G[S] \coloneqq S \cup N_G(S)$, or simply $N(S)$ and $N[S]$, respectively. For $S, T \subseteq V(G)$, we say *$S$ sees all of $T$* if $T \subseteq N(S)$.

# 3 Counting labeled chordal graphs

## 3.1 The evaporation sequence

▶ **Definition 5.** *A vertex $v$ in a graph $G$ is* simplicial *if $N(v)$ forms a clique.*

A *perfect elimination ordering* of a graph $G$ is an ordering $v_1, \ldots, v_n$ of the vertices of $G$ such that for all $i \in [n]$, $v_i$ is simplicial in the subgraph induced by the vertices $v_i, \ldots, v_n$. It is well known that a graph is chordal if and only if it has a perfect elimination ordering [5]. For our counting algorithm, we define the notion of the evaporation sequence of a chordal graph, which can be viewed as a canonical version of the perfect elimination ordering. In the evaporation sequence, rather than making an arbitrary choice for which of the simplicial vertices in $G$ will go first in the ordering, we place the set of all simplicial vertices as the first item in the sequence. As an example, if $G$ is a tree, then the set of all simplicial vertices would be exactly the leaves of $G$.

To build the evaporation sequence, we use the fact that given a chordal graph $G$, if we repeatedly remove all simplicial vertices, then eventually no vertices remain. By observing at which time step each vertex is deleted, we obtain a partition of $V(G)$, which allows us to classify and understand the structure of $G$. In our algorithm, it will also be useful to set aside a set of exceptional vertices which are never deleted, even if they are simplical.

To formalize this, suppose we are given a chordal graph $G$ and a vertex subset $X \subseteq V(G)$. We define the *evaporation sequence* of $G$ with *exception set* $X$ as follows: If $X = V(G)$, then the evaporation sequence of $G$ is the empty sequence. If $X \subsetneq V(G)$, then let $\widetilde{L}_1$ be the set of all simplicial vertices in $G$, and let $L_1 = \widetilde{L}_1 \setminus X$. Suppose $L_2, \ldots, L_t$ is the evaporation sequence of $G \setminus L_1$ (with exception set $X$). Then $L_1, L_2, \ldots, L_t$ is the evaporation sequence of $G$.

For this definition to make sense, there is one caveat: we must choose $X$ so that all vertices outside of $X$ eventually evaporate. For example, $X = \emptyset$ is always a valid choice since every chordal graph contains a simplicial vertex [5]. Without this assumption, we could potentially reach a point where $X \subsetneq V(G)$ but $X$ contains all of the simplicial vertices of $G$, in which case the evaporation sequence would not be well-defined (we never reach the base case $X = V(G)$). In our algorithm, we will always choose a valid $X$ such that all other vertices eventually evaporate.

If the evaporation sequence $L_1, L_2, \ldots, L_t$ of $G$ has length $t$, then we say $G$ *evaporates* at time $t$ with *exception set* $X$, and $t$ is called the *evaporation time*. We define $L_G(X) \coloneqq L_t$ to be the last set in the evaporation sequence of $G$, and we let $L_G(X) = \emptyset$ if the sequence is empty. Similarly, we define the evaporation time of a vertex subset: Suppose $G$ has evaporation sequence $L_1, L_2, \ldots, L_t$ with exception set $X$, and suppose $S \subseteq V(G) \setminus X$ is a nonempty vertex subset. Let $t_S$ be the largest index $i$ such that $L_i \cap S \neq \emptyset$. We say $S$ *evaporates* at time $t_S$ in $G$ with exception set $X$.

## 3.2 Setup for the counting algorithm

Given positive integers $n$ and $\omega$, we wish to count the number of $\omega$-colorable chordal graphs on $n$ vertices. This can easily be reduced to the problem of counting *connected* $\omega$-colorable chordal graphs on at most $n$ vertices (see Lemma 17 in Section 3.4). Therefore, our main focus is to describe the following algorithm:

▶ **Theorem 6.** *There is an algorithm that given $n \in \mathbb{N}$, computes the number of $\omega$-colorable labeled connected chordal graphs $G$ with vertex set $[n]$ using $O(n^7)$ arithmetic operations.*

We first give an overview of this algorithm and describe the various dynamic-programming tables (Definition 7). Next, we describe the recurrences in detail in Section 3.3. In Section 3.4, we show that the counting portion of Theorem 1 (counting chordal graphs) follows from Theorem 6 (counting connected chordal graphs). For the complete proof of Theorem 6, see the full version of the paper.

**Algorithm overview.** To count $\omega$-colorable connected chordal graphs $G$, we classify these graphs based on the behavior of their evaporation sequence. We make use of several *counter functions* (these are our dynamic-programming tables), each of which keeps track of the number of chordal graphs in a particular subclass. The arguments of the counter functions tell us the number of vertices in the graph, the evaporation time, the size of the exception set $X$, the size of the last set of simplicial vertices $L_G(X)$, etc. Initially, we consider all possibilities for the evaporation time of $G$ with exception set $X = \emptyset$. Then, using several of our recursive formulas, we reduce the number of vertices by dividing up the graph into smaller subgraphs and counting the number of possibilities for each subgraph. As we do so, the exception set $X$ increases in size. When we consider these various subgraphs, we also make sure that in each subgraph, the maximum clique size is at most $\omega$. In the end, the algorithm understands the possibilities for the entire graph, including the cliques that make up the very first set in its evaporation sequence.

The purpose of the exception set is to allow us to restrict to smaller subgraphs without distorting the evaporation behavior of the graph. For example, suppose we wish to count the number of connected chordal graphs on $n$ vertices that evaporate at time $t$, such that the vertices $1, 2, \ldots, \ell$ make up the last set to evaporate, i.e., $L_G(\emptyset) = [\ell]$. Let $L = [\ell]$. One subproblem of interest would be to count the number of possibilities for the first connected component of $G \setminus L$. Formally, we count the number of possibilities for $G' \coloneqq G[L \cup C]$, where $C$ is the connected component of $G \setminus L$ that contains the vertex $\ell + 1$. For each possible number of vertices in $G'$, we make a recursive call to count the number of possible subgraphs $G'$ of that size. However, if we were to restrict to $G'$ with a still-empty exception set, then the evaporation time of $G'$ alone could be much less than the evaporation time of $V(G')$ in $G$. Indeed, there may be vertices in $G \setminus G'$ adjacent to $L$ that prevent $L$ from evaporating before time $t$, so when we restrict to the subgraph $G'$, $L$ would now evaporate too soon. This would cause a cascading effect, causing vertices near $L$ to evaporate as well, and changing the entire evaporation sequence of $G'$. To resolve this, we add the vertices of $L$ to the exception set to preserve the evaporation behavior of $G'$.

The list of counter functions is given in Definition 7. As shown below, the number of $\omega$-colorable connected chordal graphs on $n$ vertices is the sum of various calls to the fourth function $\tilde{g}_1$, since $\tilde{g}_1(t,0,n)$ is the number of $\omega$-colorable connected chordal graphs on $n$ vertices that evaporate at time $t$ with empty exception set. To remember the names of these counter functions, one can think of them as follows: The $g$-functions keep track of the size of the exception set $X$, but these do not have information about the size of $L_G(X)$. The $f$-functions have an additional argument $\ell$, which is the size of $L_G(X)$. As a mnemonic, one can say that $g$ stands for "glued" and $f$ stands for "free." In the $g$-functions, $X$ is the "root," and all of the vertices of $X$ are glued, in the sense that they cannot evaporate. In the $f$-functions, $X \cup L_G(X)$ is the "root," and some of the vertices in the root are free, since the vertices in $L_G(X)$ are allowed to evaporate.

▶ **Definition 7.** *The following functions count particular subclasses of chordal graphs. Unless stated otherwise, the arguments $t, x, \ell, k, z$ are nonnegative integers.*

1. *$g(t,x,k,z)$ is the number of $\omega$-colorable connected chordal graphs $G$ with vertex set $[x+k]$ that evaporate in time at most $t$ with exception set $X=[x]$, where $X$ is a clique, with the following property: every connected component of $G \setminus X$ (if any) has at least one neighbor in $X \setminus [z]$.* ***Domain:*** *$t \geq 0$, $x \geq 1$, $z < x$.*
2. *$\tilde{g}(t,x,k,z)$ is the same as $g(t,x,k,z)$, except every connected component of $G \setminus X$ (if any) evaporates at time* exactly *$t$ in $G$.* Note: A graph with $V(G)=X$ would be counted because in that case, $\tilde{g}$ is the same as $g$. ***Domain:*** *$t \geq 1$, $x \geq 1$, $z < x$.*
3. *$\tilde{g}_p(t,x,k,z)$ is the same as $\tilde{g}(t,x,k,z)$, except no connected component of $G \setminus X$ sees all of $X$.* ***Domain:*** *$t \geq 1$, $x \geq 1$, $z < x$.*
4. *$\tilde{g}_1(t,x,k)$ and $\tilde{g}_{\geq 2}(t,x,k)$ are the same as $\tilde{g}(t,x,k,z)$, except every connected component of $G \setminus X$ sees all of $X$ (hence we no longer require every component of $G \setminus X$ to have a neighbor in $X \setminus [z]$), and furthermore, for $\tilde{g}_1$ we require that $G \setminus X$ has exactly one connected component, and for $\tilde{g}_{\geq 2}$ we require that $G \setminus X$ has at least two components.* ***Domain for $\tilde{g}_1$:*** *$t \geq 1$, $x \geq 0$.* ***Domain for $\tilde{g}_{\geq 2}$:*** *$t \geq 1$, $x \geq 1$.*
5. *$f(t,x,\ell,k)$ is the number of $\omega$-colorable connected chordal graphs $G$ with vertex set $[x+\ell+k]$ that evaporate at time* exactly *$t$ with exception set $X=[x]$, such that $G \setminus X$ is connected, $L_G(X)=[x+1,x+\ell]$, and $X \cup L_G(X)$ is a clique.* ***Domain:*** *$t \geq 1$, $x \geq 0$, $\ell \geq 1$.*
6. *$\tilde{f}(t,x,\ell,k)$ is the same as $f(t,x,\ell,k)$, except every connected component of $G \setminus (X \cup L_G(X))$ evaporates at time* exactly *$t-1$ in $G$, and there exists at least one such component, i.e., $X \cup L_G(X) \subsetneq V(G)$.* ***Domain:*** *$t \geq 2$, $x \geq 0$, $\ell \geq 1$.*
7. *$\tilde{f}_p(t,x,\ell,k)$ is the same as $\tilde{f}(t,x,\ell,k)$, except no connected component of $G \setminus (X \cup L_G(X))$ sees all of $X \cup L_G(X)$.* ***Domain:*** *$t \geq 2$, $x \geq 0$, $\ell \geq 1$.*
8. *$\tilde{f}_p(t,x,\ell,k,z)$ is the same as $\tilde{f}_p(t,x,\ell,k)$, except every connected component of $G \setminus (X \cup L_G(X))$ has at least one neighbor in $(X \cup L_G(X)) \setminus [z]$.* ***Domain:*** *$t \geq 2$, $x \geq 0$, $\ell \geq 1$, $z \leq x$.*

## 3.3 Recurrences for the counting algorithm

We implicitly assume *all graphs in this section are connected and $\omega$-colorable.* For $k \in \mathbb{N}$, let $c(k)$ denote the number of $\omega$-colorable connected chordal graphs with vertex set $[k]$. To compute $c(n)$, we first consider all possibilities for the evaporation time. Initially, the exception set is empty. We observe that $\tilde{g}_1(t,0,n)$ is the number of (connected, $\omega$-colorable) chordal graphs with vertex set $[n]$ that evaporate at time exactly $t$ with empty exception set.

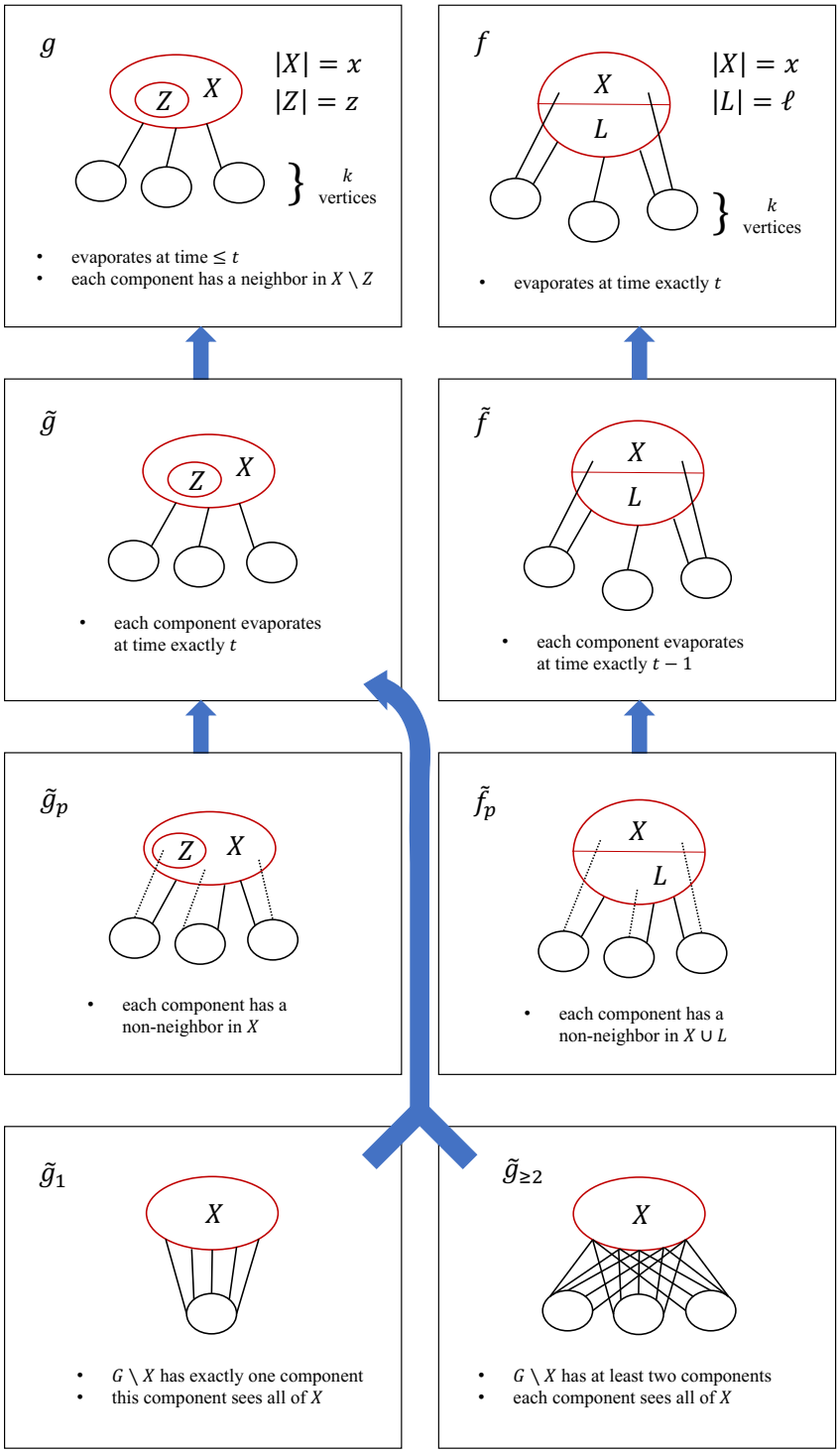



**Figure 1** The counter functions. Here $L = L_G(X)$ and $Z = [z]$. An arrow from one function to another, say from $\tilde{g}$ to $g$, indicates that the definition of $\tilde{g}$ is the same as that of $g$, except where indicated otherwise. The drawing of $\tilde{f}_p$ represents $\tilde{f}_p(t, x, \ell, k)$. The function $\tilde{f}_p(t, x, \ell, k, z)$ is similar but also keeps track of the argument $z$.

Therefore,

$$c(n) = \sum_{t=1}^{n} \tilde{g}_1(t, 0, n).$$

We compute the necessary values of $\tilde{g}_1$ by evaluating the following recurrences top-down using memoization. We take this approach rather than bottom-up dynamic programming to simplify the description slightly, since by memoizing we do not need to specify in what order the entries of the various dynamic-programming tables are computed. We simply compute each value of the counter functions as needed. For the following recurrences, let $X = [x]$ according to the current value of the argument $x$, and let $L = L_G(X)$.

To compute $\tilde{g}_1(t, 0, n)$, the number of chordal graphs that evaporate at time exactly $t$, we consider all possibilities for the size $\ell$ of $L$. Once the size $\ell$ is given, there are $\binom{n}{\ell}$ possibilities for the label set of $L$. Recall that $f$ counts the number of chordal graphs where $L$ is fixed and evaporates at time $t$, and $G \setminus X$ is connected. Formally, we have the following recurrence – the first time this is used, the arguments are $t$, $x = 0$ and $k = n$.

▶ **Lemma 8.** *For $\tilde{g}_1$, we have*

$$\tilde{g}_1(t, x, k) = \sum_{\ell=1}^{k} \binom{k}{\ell} f(t, x, \ell, k - \ell).$$

The proof of Lemma 8, along with the proofs of all of the other recurrences, can be found in the full version of the paper. To see the intuition behind Lemma 8, recall that in the definition of $f(t, x, \ell, k)$ we require a specific label set for $L_G(X)$, namely $[x + 1, x + \ell]$. If we were to replace that requirement with $L_G(X) = L'$ for any other subset $L'$ of $[x+1, x+\ell+k]$ of size $\ell$, this would not change the value of $f(t, x, \ell, k)$. Therefore, in the recurrence for $\tilde{g}_1$, it is sufficient to compute $f(t, x, \ell, k - \ell)$ and multiply by $\binom{k}{\ell}$, rather than computing $\binom{k}{\ell}$ distinct counter functions.

For $f$, to count chordal graphs where $L$ is fixed and evaporates at time $t$, and $G \setminus X$ is connected, we consider all possibilities for the set of labels that appear in components of $G \setminus (X \cup L)$ that evaporate at time exactly $t - 1$. Recall that $\tilde{f}$ counts the number of chordal graphs where $L$ is fixed and evaporates at time $t$, $G \setminus X$ is connected, and all components of $G \setminus (X \cup L)$ evaporate at time exactly $t - 1$. For each possible $k'$ (the size of this label set), $\tilde{f}$ allows us to count the number of possibilities for the subgraph $G_1$ consisting of $X \cup L$ and all components of $G \setminus (X \cup L)$ that evaporate at time exactly $t - 1$, and $g$ allows us to count the number of possibilities for the subgraph $G_2$ consisting of $X \cup L$ and all other components.

▶ **Lemma 9.** *For $f$, we have*

$$f(t, x, \ell, k) = \sum_{k'=1}^{k} \binom{k}{k'} \tilde{f}(t, x, \ell, k') g(t - 2, x + \ell, k - k', x).$$

When $f$ is called for the first time in the initial steps of the algorithm, this is the first moment when $X$ becomes nonempty, since at this point we are restricting to a subgraph with fewer than $n$ vertices. When we restrict to the subgraph $G_2$, we want to ensure that its vertices have the same evaporation behavior as they did in $G$. In particular, we need to ensure that the vertices of $L$ do not evaporate too soon, since their presence may be essential for preventing other vertices from evaporating. For this reason, we let $X \cup L$ be the exception set for $G_2$. For $G_1$, the exception set is simply $X$ because the components that evaporate at time exactly $t - 1$ are still present in $G_1$, preventing the vertices of $L$ from evaporating before time $t$.

For the subgraph $G_2$, now that all of $L$ has been pushed into the exception set, we no longer have information about the argument $\ell$ since the last set of simplicial vertices in $G_2$ evaporates further back in time. This is why we call $g$ rather than $f$ to count the possibilities for $G_2$. In fact, $G_2 \setminus [x + \ell]$ might not even be connected, which is required by $f$. Finally, the fourth argument of $g$ indicates that every connected component of $G_2 \setminus (X \cup L)$ has at least one neighbor in $L$. This ensures that $G \setminus X$ is connected.

For $g$, to count chordal graphs that evaporate in time at most $t$, we consider all possibilities for the set of labels that appear in connected components of $G \setminus X$ that evaporate at time exactly $t$. Recall that $\tilde{g}$ counts the number of chordal graphs where all connected components of $G \setminus X$ evaporate at time exactly $t$.

▶ **Lemma 10.** *For $g$, we have*

$$g(t, x, k, z) = \sum_{k'=0}^{k} \binom{k}{k'} \tilde{g}(t, x, k', z) g(t-1, x, k-k', z).$$

For $\tilde{g}$, to count chordal graphs where all connected components of $G \setminus X$ evaporate at time exactly $t$, we consider all possible label sets for the component $C$ of $G \setminus X$ that contains the lowest label not in $X$. The constraint $x' \geq 1$ ensures that $G$ is connected. We also subtract all ways of selecting $x'$ elements from $[z]$ to ensure that $N(C)$ is not entirely contained in $[z]$.

▶ **Lemma 11.** *For $\tilde{g}$, we have*

$$\tilde{g}(t, x, k, z) = \sum_{k'=1}^{k} \sum_{x'=1}^{x} \left( \binom{x}{x'} - \binom{z}{x'} \right) \binom{k-1}{k'-1} \tilde{g}_1(t, x', k') \tilde{g}(t, x, k-k', z).$$

We subtract 1 in the binomial coefficient $\binom{k-1}{k'-1}$ because the label set for $C$ always contains the lowest non-$X$ label, along with $k' - 1$ other labels.

For $\tilde{f}$, we need to count chordal graphs where $L$ is fixed and evaporates at time $t$, $G \setminus X$ is connected, and all components of $G \setminus (X \cup L)$ evaporate at time exactly $t-1$. The number of such graphs in which zero components see all of $X \cup L$ is $\tilde{f}_p(t, x, \ell, k)$. Now if there is at least one all-seeing component, then we break this down into two further cases: either exactly one component sees all of $X \cup L$, or at least two components see all of $X \cup L$. Recall that $\tilde{f}_p$ counts the number of chordal graphs where $L$ is fixed and evaporates at time $t$, all components of $G \setminus (X \cup L)$ evaporate at time exactly $t-1$, and no component sees all of $X \cup L$. Also, recall that $\tilde{g}_{\geq 2}$ counts the number of chordal graphs where all components of $G \setminus X$ evaporate at time exactly $t$, every component sees all of $X$, and there are at least two such components. In the first (resp. second) case, $\tilde{g}_1$ (resp. $\tilde{g}_{\geq 2}$) corresponds to the all-seeing component(s), and $\tilde{f}_p$ (resp. $\tilde{g}_p$) corresponds to the remaining components.

▶ **Lemma 12.** *For $\tilde{f}$, we have*

$$\begin{aligned}\tilde{f}(t, x, \ell, k) = \tilde{f}_p(t, x, \ell, k) &+ \sum_{k'=1}^{k} \binom{k}{k'} \tilde{g}_1(t-1, x+\ell, k') \tilde{f}_p(t, x, \ell, k-k') \\ &+ \sum_{k'=1}^{k} \binom{k}{k'} \tilde{g}_{\geq 2}(t-1, x+\ell, k') \tilde{g}_p(t-1, x+\ell, k-k', x).\end{aligned}$$

The above cases are relevant because if at least two components of $G \setminus (X \cup L)$ see all of $X \cup L$, then this prevents the vertices of $L$ from evaporating before time $t$. Indeed, each vertex $u \in L$ has a neighbor in each of those two components, meaning $u$ has two non-adjacent neighbors. Otherwise, if there is at most one such component, then the neighborhoods of

the remaining components of $G \setminus (X \cup L)$ must together cover $L$ to ensure that $L$ does not evaporate until time $t$. In that case, for each vertex $u \in L$ that is covered by a proper-subset neighborhood $N(C)$ of a component $C$, $u$ has a neighbor $v \in C$ as well as a neighbor $w \in (X \cup L) \setminus N(C)$, and $v$ and $w$ are non-adjacent.

The reason we require $G \setminus X$ to be connected in the definition of $f$ (rather than just requiring $G$ to be connected) can be seen from the recurrence for $\tilde{f}$. Since in the first sum over $k'$ we only wish to consider graphs with exactly one all-seeing component, in the definition of $\tilde{g}_1$ we require $G \setminus X$ to be connected. The recurrence for $\tilde{g}_1$ depends on $f$, so this carries over into requiring $G \setminus X$ to be connected in the definition of $f$. This explains the need for the argument $z$ (for example, in $g$): as mentioned above, keeping track of $z$ lets us ensure that $G \setminus X$ is connected in all graphs counted by $f$.

For $\tilde{g}_{\geq 2}$, to count chordal graphs where all components of $G \setminus X$ evaporate at time exactly $t$, every component sees all of $X$, and there are at least two such components, we consider all possibilities for the label set of the component that contains the lowest label not in $X$. For the remaining components, there is either exactly one of them, or at least two.

▶ **Lemma 13.** *For $\tilde{g}_{\geq 2}$, we have*

$$\tilde{g}_{\geq 2}(t,x,k) = \sum_{k'=1}^{k-1} \binom{k-1}{k'-1} \tilde{g}_1(t,x,k') \Big( \tilde{g}_1(t,x,k-k') + \tilde{g}_{\geq 2}(t,x,k-k') \Big).$$

For $\tilde{g}_p$, to count chordal graphs where all components of $G \setminus X$ evaporate at time exactly $t$ and no component sees all of $X$, we proceed as we did for $\tilde{g}$, except we require $x' < x$ rather than $x' \leq x$.

▶ **Lemma 14.** *For $\tilde{g}_p$, we have*

$$\tilde{g}_p(t,x,k,z) = \sum_{k'=1}^{k} \sum_{x'=1}^{x-1} \left( \binom{x}{x'} - \binom{z}{x'} \right) \binom{k-1}{k'-1} \tilde{g}_1(t,x',k') \tilde{g}_p(t,x,k-k',z).$$

For $\tilde{f}_p$, to count chordal graphs where $L$ is fixed and evaporates at time $t$, all components of $G \setminus (X \cup L)$ evaporate at time exactly $t-1$, and no component sees all of $X \cup L$, we first declare that no component can see only into $X$ (since $G \setminus X$ is connected).

▶ **Lemma 15.** *We have $\tilde{f}_p(t,x,\ell,k) = \tilde{f}_p(t,x,\ell,k,x)$.*

The following recurrence for $\tilde{f}_p$ counts the number of such graphs in which every component of $G \setminus (X \cup L)$ has at least one neighbor in $(X \cup L) \setminus [z]$. On the first reading, one can skip the two "otherwise" cases in Lemma 16. In this lemma, we consider all possibilities for the label set of the component $C$ of $G \setminus (X \cup L)$ that contains the lowest label not in $X \cup L$. Additionally, we consider all possibilities for the size $x'$ of $N(C) \cap X$ and the size $\ell'$ of $N(C) \cap L$, and we consider all possibilities for their respective label sets. If $\ell' > 0$, then $N(C)$ is automatically not contained in $[z]$ since $z \leq x$, so there are $\binom{x}{x'}$ possible label sets for $N(C) \cap X$.

The intuition behind the two "otherwise" cases is as follows. If $\ell' = 0$, then we must subtract $\binom{z}{x'}$ from the number of possible label sets for $N(C) \cap X$ to ensure that $N(C) \subsetneq [z]$. If $\ell' = \ell$, then all of the vertices of $L$ have now been pushed into the exception set, so the evaporation time of the subgraph formed from the remaining components is $t-1$. In this case, we call $\tilde{g}_p$ since we no longer know the size of the last set of simplicial vertices.[1]

[1] One might wonder whether we depart from the domain of $\tilde{g}_p$ in the term $\tilde{g}_p(t-1, x+\ell', k-k', z)$, since $x + \ell' = z$ when $\ell' = 0$ and $x = z$. However, if $\ell' = 0$ and $x = z$, then we observe that $\binom{x}{x'} - \binom{z}{x'} = 0$. Thus, for this value of $\ell'$, we do not need to evaluate the calls to $\tilde{g}_1$, $\tilde{f}_p$, and $\tilde{g}_p$.

▶ **Lemma 16.** *For $\tilde{f}_p(t,x,\ell,k,z)$, we have*

$$\tilde{f}_p(t,x,\ell,k,z) = \sum_{k'=1}^{k} \sum_{\substack{0\le x'\le x\\ 0\le \ell'\le \ell\\ 0<x'+\ell'<x+\ell}} \binom{k-1}{k'-1}\binom{\ell}{\ell'}\tilde{g}_1(t-1,x'+\ell',k') \cdot \begin{cases} \binom{x}{x'} & \text{if } \ell'>0 \\ \binom{x}{x'}-\binom{z}{x'} & \text{otherwise} \end{cases} \cdot \begin{cases} \tilde{f}_p(t,x+\ell',\ell-\ell',k-k',z) & \text{if } \ell'<\ell \\ \tilde{g}_p(t-1,x+\ell',k-k',z) & \text{otherwise.} \end{cases}$$

The base cases are as follows. We reach the base case for $g$ when $t=0$:

$$g(0,x,k,z) = \begin{cases} 1 & \text{if } k=0 \\ 0 & \text{if } k>0. \end{cases}$$

For $\tilde{g}$ and $\tilde{g}_p$, we have $\tilde{g}(t,x,0,z)=1$ and $\tilde{g}_p(t,x,0,z)=1$ when $k=0$. For, $\tilde{g}_1$ we observe that $\tilde{g}_1(t,x,k)=0$ if $t=0$ or $k=0$. Similarly, for $\tilde{g}_{\ge 2}$ we have $\tilde{g}_{\ge 2}(t,x,k)=0$ if $t=0$ or $k=0$. We reach the base case for $f$ when $x+\ell>\omega$, $t=1$, or $k=0$. If $x+\ell>\omega$, then $f(t,x,\ell,k)=0$. Remarkably, this is the only place where $\omega$ appears in the algorithm. If $x+\ell\le\omega$, then we have

$$f(1,x,\ell,k) = \begin{cases} 1 & \text{if } k=0 \\ 0 & \text{otherwise.} \end{cases}$$

If $x+\ell\le\omega$ and $t\ge 2$, then $f(t,x,\ell,0)=0$. For $\tilde{f}$, we have $\tilde{f}(t,x,\ell,k)=0$ if $t=1$ or $k=0$. Similarly, for $\tilde{f}_p$ we have $\tilde{f}_p[t,x,\ell,k,z]=0$ if $t=1$ or $k=0$. For the version of $\tilde{f}_p$ without the fifth argument $z$, we do not need a base case since we always immediately call $\tilde{f}_p$ with $z$.

The control flow formed by these recurrences is shown in Figure 2. The algorithm terminates because either the value of $t$ or the number of vertices in the graph (i.e., $x+k$ or $x+\ell+k$) decreases each time we return to the same function. For the running time, this is dominated by the arithmetic operations needed to compute $\tilde{f}_p$. The recurrence for $\tilde{f}_p$ involves a triple summation, and there are five arguments, so a naive implementation uses $O(n^8)$ arithmetic operations. However, in the full version of the paper, we show that the running time can in fact be improved to $O(n^7)$ arithmetic operations.

## 3.4 Proof of Theorem 1 (counting)

In this section, we prove the counting portion of Theorem 1 using Theorem 6. (See the full version of the paper for the proof of the sampling portion of Theorem 1.) In other words, we describe an algorithm to count chordal graphs, assuming we have an algorithm to count connected chordal graphs. Theorem 6 – counting connected chordal graphs – is proved in the full version of the paper.

For $k\in\mathbb{N}$, let $a(k)$ denote the number of $\omega$-colorable chordal graphs with vertex set $[k]$. Recall that $c(k)$ is the number of $\omega$-colorable connected chordal graphs with vertex set $[k]$.

▶ **Lemma 17.** *The number of $\omega$-colorable chordal graphs with vertex set $[n]$ is given by*

$$a(n) = \sum_{k=1}^{n} \binom{n-1}{k-1} c(k)a(n-k)$$

*for all $n\in\mathbb{N}$.*

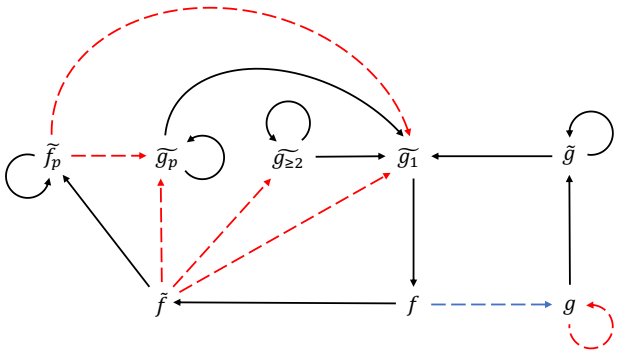



**Figure 2** The control flow of the counting algorithm. An arrow from $f$ to $g$ indicates that the recursive formula for $f$ depends on $g$. The arrow is red (and dashed) if $t$ decreases by 1, blue (and dashed) if $t$ decreases by 2, and black if $t$ does not change. The black self-loops do not cause an infinite loop because in those recursive calls, the number of vertices decreases.

**Proof.** Suppose $G$ is an $\omega$-colorable chordal graph with vertex set $[n]$. Let $G_1$ be the graph formed by the connected component of $G$ that contains the label 1, and let $G'$ be the graph formed by all other connected components of $G$ (which can potentially be empty). Let $C$ be the set of labels that appear in $G_1$, let $k = |C|$, and let $D = [n] \setminus C$, i.e., $D$ is the set of labels that appear in $G'$. Now relabel $G_1$ by applying $\phi(C, [k])$ to the labels in $C$, and relabel $G'$ by applying $\phi(D, [n-k])$ to the labels in $D$. (Recall that $\phi(A, B)$ is defined in Section 2.) We can see that $G_1$ is now a connected $\omega$-colorable chordal graph with vertex set $[k]$, and $G'$ is a connected $\omega$-colorable chordal graph with vertex set $[n-k]$. The map that takes any chordal graph $G$ to the resulting pair $(G_1, G')$ is injective since $\phi(C, [k])$ and $\phi(D, [n-k])$ are both bijections. Therefore, $a(n)$ is at most the number of possible triples $(G_1, G', C)$, which is given by the summation above.

To see that $a(n)$ is bounded below by the same summation, suppose we are given $1 \leq k \leq n$, a connected $\omega$-colorable chordal graph $G_1$ with vertex set $[k]$, a chordal $\omega$-colorable graph $G'$ with vertex set $[n-k]$, and a subset $C \subseteq [n]$ of size $k$ that contains 1. Let $D = [n] \setminus C$. We construct an $\omega$-colorable chordal graph $G$ with vertex set $[n]$ as follows: Relabel $G_1$ by applying $\phi([k], C)$ to its label set, and relabel $G'$ by applying $\phi([n-k], D)$ to its label set. Now let $G$ be the union of $G_1$ and $G'$ (by taking the union of the vertex sets and the edge sets). The map that takes the triple $(G_1, G', C)$ to the resulting graph $G$ is injective, so $a(n)$ is at least the summation above, as desired. ◀

Lemma 17 directly gives a dynamic-programming algorithm to compute the number of $\omega$-colorable chordal graphs with vertex set $[n]$, given $n$ as input. First, by Theorem 6, we can compute $c(k)$ for all $k \in [n]$ at a cost of $O(n^7)$ arithmetic operations. Next, we use the recurrence in Lemma 17 to compute $a(n)$ at a cost of $O(n^2)$ arithmetic operations. For the base case, we observe that $a(0) = 1$. Therefore, we have an algorithm to count $\omega$-colorable chordal graphs on $n$ vertices using $O(n^7)$ arithmetic operations.

## 4 Implementation of the counting algorithm

An implementation of the counting algorithm in C++ can be successfully run for inputs as large as $n = 30$ in about 2.5 minutes on a standard desktop computer.[2] Previously, the number of labeled chordal graphs was only known up to $n = 15$. Table 1 shows the number of

[2] Our implementation is available on GitHub at `https://github.com/uhebertj/chordal`.

connected chordal graphs on $n$ vertices for $n \leq 30$, with the chromatic number unrestricted. Table 2 shows the number of $\omega$-colorable connected chordal graphs on $n$ vertices for various values of $n$ and $\omega$.

**Table 1** Numbers of labeled connected chordal graphs on $n$ vertices.

| $c(n)$ | $n$ |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 4 | 3 |
| 35 | 4 |
| 541 | 5 |
| 13302 | 6 |
| 489287 | 7 |
| 25864897 | 8 |
| 1910753782 | 9 |
| 193328835393 | 10 |
| 26404671468121 | 11 |
| 4818917841228328 | 12 |
| 1167442027829857677 | 13 |
| 374059462390709800421 | 14 |
| 158311620026439080777076 | 15 |
| 88561607724193506845709239 | 16 |
| 65629642803250494352023169033 | 17 |
| 64646285130595946195244365518454 | 18 |
| 84997214469704246545711429635276299 | 19 |
| 14988142356875294544461626191310904642 1 | 20 |
| 35626055123928426690872494367291110048855 8 | 21 |
| 11473744949464491944508258176053401236791504 61 | 22 |
| 50324868520402653224615508446959396780529673840 53 | 23 |
| 30210545039307528599583618386687349227933725131035504 | 24 |
| 24940038313065905058019326786145957925448982265006568596 1 | 25 |
| 28441345486995689815615546290431460703243324009448674823403 13 | 26 |
| 44993294034522185332489548856700572371349354518671249097245374660 | 27 |
| 99127725139236030144346028839700910906670827577808606147000987702773 9 | 28 |
| 30526157144572224953157514915475479605501638476250575941226904780179348933 | 29 |
| 13183638007395954271288355542312707702094261964027362487431622588244921589952 54 | 30 |

## 5 Conclusion

Our main result is an algorithm that given $n$, computes the number of labeled chordal graphs on $n$ vertices using $O(n^7)$ arithmetic operations (and in $O(n^9 \log n)$ time in the RAM model). This yields a sampling algorithm that generates a labeled chordal graph on $n$ vertices uniformly at random. For the sampling algorithm, once we have run the counting algorithm as a preprocessing step, each sample can be obtained using $O(n^4)$ arithmetic operations.

The main open problem is to design a substantially faster algorithm for counting or sampling labeled chordal graphs. We presented exact counting and sampling algorithms; nevertheless, allowing for approximate counting/sampling might enable even faster algorithms.

**Table 2** Numbers of $\omega$-colorable labeled connected chordal graphs on $n$ vertices. When $\omega = 2$, the algorithm counts labeled trees.

| | | | | | $n$ | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
| 1 | 3 | 16 | 125 | 1296 | 16807 | 262144 | 4782969 | 2 |
| | 4 | 34 | 480 | 9831 | 268093 | 9185436 | 379623492 | 3 |
| | | 35 | 540 | 13136 | 466683 | 22732032 | 1437072780 | 4 |
| | | | 541 | 13301 | 488873 | 25736782 | 1873146621 | 5 |
| | | | | 13302 | 489286 | 25863916 | 1910084529 | 6 |
| | | | | | 489287 | 25864896 | 1910751531 | 7 |
| | | | | | | 25864897 | 1910753781 | 8 |
| | | | | | | | 1910753782 | 9 |

$\omega$

| | | $n$ | | |
|---|---|---|---|---|
| 10 | 11 | 12 | | |
| 100000000 | 2357947691 | 61917364224 | 2 | |
| 18376225525 | 1019282908941 | 63707908718994 | 3 | |
| 112588153700 | 10535042533301 | 1144261607209084 | 4 | |
| 181962472490 | 22726623077466 | 3513611793935959 | 5 | |
| 192919501307 | 26158547399061 | 4666697716137194 | 6 | |
| 193325509217 | 26400465973728 | 4813890013657154 | 7 | $\omega$ |
| 193328830337 | 26404655450778 | 4818876084111431 | 8 | |
| 193328835392 | 26404671456933 | 4818917765689886 | 9 | |
| 193328835393 | 26404671468120 | 4818917841203841 | 10 | |
| | 26404671468121 | 4818917841228327 | 11 | |
| | | 4818917841228328 | 12 | |

To be precise, for approximate sampling we are aiming for an algorithm that, given $n$ and $\delta > 0$, samples from a distribution $\delta$-close to uniform (say in total variation distance) in time polynomial in $n$ and $\log(1/\delta)$, where the dependence on $n$ is significantly less than $n^7$. Two interesting approaches to consider are Markov Chain Monte Carlo (MCMC) algorithms, such as the chain proposed in [33], and the Boltzmann sampling scheme used in [10] for planar graphs.

Moving beyond chordal graphs, there are many interesting graph classes for which the problem of counting/sampling $n$-vertex labeled graphs in polynomial time appears to be open, including perfect graphs, weakly chordal graphs, strongly chordal graphs, and chordal bipartite graphs, as well as many graph classes characterized by a finite set of forbidden minors, subgraphs, or induced subgraphs. It is worth noting that for some well-known graph classes of this form, such as planar graphs, polynomial-time algorithms are known [6, 10].

## References


1 Andrii Arman, Pu Gao, and Nicholas Wormald. Fast uniform generation of random graphs with given degree sequences. *Random Structures & Algorithms*, 59(3):291–314, 2021.

2 EA Bender, LB Richmond, and NC Wormald. Almost all chordal graphs split. *Journal of the Australian Mathematical Society*, 38(2):214–221, 1985.

3 Ivona Bezáková, Nayantara Bhatnagar, and Eric Vigoda. Sampling binary contingency tables with a greedy start. *Random Structures & Algorithms*, 30(1-2):168–205, 2007.

4 Ivona Bezáková and Wenbo Sun. Mixing of Markov chains for independent sets on chordal graphs with bounded separators. In *International Computing and Combinatorics Conference (COCOON)*, pages 664–676, 2020.

5 Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

6 Manuel Bodirsky, Clemens Gröpl, and Mihyun Kang. Generating labeled planar graphs uniformly at random. *Theoretical Computer Science*, 379(3):377–386, 2007.

7 Andreas Brandstädt, Van Bang Le, and Jeremy P Spinrad. *Graph classes: a survey*. SIAM, 1999.

8 Tınaz Ekim, Mordechai Shalom, and Oylum Şeker. The complexity of subtree intersection representation of chordal graphs and linear time chordal graph generation. *J. Comb. Optim.*, 41(3):710–735, 2021.

9 Martin Farber. Independent domination in chordal graphs. *Operations Research Letters*, 1(4):134–138, 1982.

10 Éric Fusy. Uniform random sampling of planar graphs in linear time. *Random Structures & Algorithms*, 35(4):464–522, 2009.

11 Pu Gao and Nicholas Wormald. Uniform generation of random regular graphs. *SIAM Journal on Computing*, 46:1395–1427, 2017.

12 Pu Gao and Nicholas Wormald. Uniform generation of random graphs with power-law degree sequences. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1741–1758, 2018.

13 Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, 1972.

14 Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

15 Catherine Greenhill. *Generating graphs randomly*, pages 133–186. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021.

16 Catherine Greenhill and Matteo Sfragara. The switch Markov chain for sampling irregular graphs and digraphs. *Theoretical Computer Science*, 719:1–20, 2018.

17 Dan Gusfield. The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. In *Research in Computational Molecular Biology (RECOMB)*, pages 236–252, 2009.

18 András Hajnal and János Surányi. Über die auflösung von graphen in vollständige teilgraphen. *Ann. Univ. Sci. Budapest, Eötvös Sect. Math*, 1:113–121, 1958.

19 David Harvey and Joris Van Der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

20 Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

21 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *Journal of the ACM*, 51(4):671–697, 2004.

22 Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

23 Lilian Markenzon, Oswaldo Vernet, and Luiz Henrique Araujo. Two methods for the generation of chordal graphs. *Annals of Operations Research*, 157:47–60, 2008.

24 Asish Mukhopadhyay and Md. Zamilur Rahman. Algorithms for generating strongly chordal graphs. In *Transactions on Computational Science XXXVIII*, pages 54–75, 2021.

25 OEIS Foundation Inc. Entry A058862 in the On-Line Encyclopedia of Integer Sequences, 2023. Published electronically at `http://oeis.org`.

**26** Yoshio Okamoto, Takeaki Uno, and Ryuhei Uehara. Counting the number of independent sets in chordal graphs. *J. Discrete Algorithms*, 6(2):229–242, 2008.

**27** Heinz Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

**28** Teresa Przytycka, George Davis, Nan Song, and Dannie Durand. Graph theoretical insights into evolution of multidomain proteins. *J Comput Biol.*, 13(2):351–363, 2006.

**29** H.N. de Ridder et al. Information system on graph classes and their inclusions (ISGCI). *www.graphclasses.org*, 2016.

**30** Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph theory and computing*, pages 183–217. Elsevier, 1972.

**31** Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

**32** Oylum Şeker, Pinar Heggernes, Tınaz Ekim, and Z. Caner Taşkın. Linear-time generation of random chordal graphs. In *Proceedings of the 10th International Conference on Algorithms and Complexity (CIAC)*, pages 442–453, 2017.

**33** Wenbo Sun and Ivona Bezáková. Sampling random chordal graphs by MCMC (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34(10), pages 13929–13930, 2020.

**34** Marcel Wienöbst, Max Bannach, and Maciej Liśkiewicz. Polynomial-time algorithms for counting and sampling Markov equivalent DAGs. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, pages 12198–12206, 2021.

**35** Nicholas C. Wormald. Counting labelled chordal graphs. *Graphs and Combinatorics*, 1:193–200, 1985.

**36** Nicholas C. Wormald. Generating random unlabelled graphs. *SIAM Journal on Computing*, 16(4):717–727, 1987.

**37** Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

# Tight Algorithms for Connectivity Problems Parameterized by Clique-Width

**Falko Hegerfeld**
Humboldt-Universität zu Berlin, Germany

**Stefan Kratsch**
Humboldt-Universität zu Berlin, Germany

## Abstract

The complexity of problems involving global constraints is usually much more difficult to understand than the complexity of problems only involving local constraints. In the realm of graph problems, connectivity constraints are a natural form of global constraints. We study connectivity problems from a fine-grained parameterized perspective. In a breakthrough result, Cygan et al. (TALG 2022) first obtained Monte-Carlo algorithms with single-exponential running time $\alpha^{\mathrm{tw}}n^{\mathcal{O}(1)}$ for connectivity problems parameterized by treewidth by introducing the cut-and-count-technique, which reduces many connectivity problems to locally checkable counting problems. Furthermore, the obtained bases $\alpha$ were shown to be optimal under the Strong Exponential-Time Hypothesis (SETH).

However, since only sparse graphs may admit small treewidth, we lack knowledge of the fine-grained complexity of connectivity problems with respect to dense structure. The most popular graph parameter to measure dense structure is arguably clique-width, which intuitively measures how easily a graph can be constructed by repeatedly adding bicliques. Bergougnoux and Kanté (TCS 2019) have shown, using the rank-based approach, that also parameterized by clique-width many connectivity problems admit single-exponential algorithms. Unfortunately, the obtained running times are far from optimal under SETH.

We show how to obtain optimal running times parameterized by clique-width for two benchmark connectivity problems, namely Connected Vertex Cover and Connected Dominating Set. These are the first tight results for connectivity problems with respect to clique-width and these results are obtained by developing new algorithms based on the cut-and-count-technique and novel lower bound constructions. Precisely, we show that there exist one-sided error Monte-Carlo algorithms that given a $k$-clique-expression solve

- Connected Vertex Cover in time $6^k n^{\mathcal{O}(1)}$, and
- Connected Dominating Set in time $5^k n^{\mathcal{O}(1)}$.

Both results are shown to be tight under SETH.

**2012 ACM Subject Classification** Mathematics of computing → Paths and connectivity problems; Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Combinatorial algorithms

**Keywords and phrases** Parameterized Complexity, Connectivity, Clique-width, Cut&Count, Lower Bound

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.59

**Related Version** *Full Version*: https://arxiv.org/abs/2302.03627

**Funding** *Falko Hegerfeld*: Partially supported by DFG Emmy Noether-grant (KR 4286/1).

## 1 Introduction

One way to cope with the NP-hardness of a problem is the theory of parameterized complexity, where we seek to solve structured instances faster than worst-case instances; an additional parameter quantifies how structured an instance is. Ideally, we obtain fixed-parameter


tractable algorithms with running time[1] $\mathcal{O}^*(f(k))$, where $k$ is the parameter and $f$ some computable function. Having established the existence of such an algorithm, the next natural step is to take a fine-grained perspective and to determine the smallest possible function $f$, which quantifies the precise impact of the considered structure on problem complexity.

We apply this approach to connectivity problems. Our investigation starts with the breakthrough results of Cygan et al. [16], who for the first time obtained Monte-Carlo algorithms with running time $\mathcal{O}^*(\alpha^{\mathrm{tw}})$, for some constant *base* $\alpha > 1$, for connectivity problems parameterized by *treewidth* (tw). These algorithms are obtained via the so-called cut-and-count-technique, which reduces connectivity problems to locally checkable counting problems. In addition, the obtained bases $\alpha$ were proven to be optimal assuming the Strong Exponential-Time Hypothesis (SETH) [14].

As only sparse graphs may have small treewidth, we lack knowledge of the precise complexity of connectivity problems with respect to dense structure. In the regime of dense graphs, *clique-width* (cw) is one of the most popular parameters. Bergougnoux [2] applied cut-and-count to several width-parameters based on structured neighborhoods with clique-width among these. Moreover, Bergougnoux and Kanté [4], building upon the rank-based approach of Bodlaender et al. [8], obtain single-exponential running times $\mathcal{O}^*(\alpha^{\mathrm{cw}})$ for a large class of connectivity problems parameterized by clique-width. As both articles are aimed at obtaining a breadth of single-exponential algorithms for a large class of problems, the Connected (Co-)$(\sigma,\rho)$-Dominating Set problems, the obtained bases for particular problems are far from being optimal. For example, the former article implies an $\mathcal{O}^*(128^{\mathrm{cw}})$-time algorithm for Connected Dominating Set and the latter yields an $\mathcal{O}^*((27 \cdot 2^{\omega+1})^{\mathrm{cw}})$-time algorithm for Connected Vertex Cover and an $\mathcal{O}^*((8 \cdot 2^{\omega+1})^{\mathrm{cw}})$-time algorithm for Connected Dominating Set, where $\omega$ is the matrix multiplication exponent, see e.g. Alman and Vassilevska W. [1]. Even if $\omega = 2$, this only yields the large bases 216 and 64 respectively.

We show that the running times for Connected Vertex Cover and Connected Dominating Set parameterized by clique-width can be considerably optimized by providing novel algorithms. These faster algorithms again rely on the cut-and-count-technique and are fine-tuned by precisely analyzing which cut-and-count states are necessary to consider. Moreover, we use further techniques such as fast subset convolution, inclusion-exclusion states, and distinguishing between live and dead labels to obtain the improved running times.

▶ **Theorem 1.1.** *There are one-sided error Monte-Carlo algorithms that, given a $k$-expression[2] for a graph $G$, can solve*

- *Connected Vertex Cover in time $\mathcal{O}^*(6^k)$,*
- *Connected Dominating Set in time $\mathcal{O}^*(5^k)$.*

We show that these algorithms are essentially the correct ones for these problem-parameter-combinations by proving that the obtained running times are optimal under SETH. To prove these lower bounds, we follow the by now standard construction principle of Lokshtanov et al. [35] for lower bounds relative to width-parameters. To apply this principle for clique-width, we closely investigate the problem behavior across *joins*, i.e., the edge-structures via which clique-width is defined, and the results of this investigation strongly guide us in designing appropriate gadgets. Precisely, we obtain the following tight lower bounds:

[1] The $\mathcal{O}^*$-notation hides polynomial factors in the input size.
[2] A $k$-expression witnesses that the clique-width of $G$ is at most $k$.

**Table 1** Optimal running-times of several connectivity problems with respect to various width-parameters listed in increasing generality. The results in the last column are obtained in this paper. Between modular-treewidth and clique-width, we only have the relationship $\text{cw}(G) \leq 3 \cdot 2^{\text{mod-tw}(G)-1}$, but the same results are also tight for the more restrictive modular-pathwidth, where we have $\text{cw}(G) \leq \text{mod-pw}(G) + 2$ by Hegerfeld and Kratsch [26]. The "?" marks problem-parameter combinations, where an algorithm with single-exponential running time is known by Bergougnoux and Kanté [4], but a gap between the lower bound and algorithm remains.

| Parameters | cutwidth | treewidth | modular-tw | clique-width |
|---|---|---|---|---|
| Connected Vertex Cover | $\mathcal{O}^*(2^k)$ | $\mathcal{O}^*(3^k)$ | $\mathcal{O}^*(5^k)$ | $\mathcal{O}^*(6^k)$ |
| Connected Dominating Set | $\mathcal{O}^*(3^k)$ | $\mathcal{O}^*(4^k)$ | $\mathcal{O}^*(4^k)$ | $\mathcal{O}^*(5^k)$ |
| Steiner Tree | $\mathcal{O}^*(3^k)$ | $\mathcal{O}^*(3^k)$ | $\mathcal{O}^*(3^k)$ | ? |
| Feedback Vertex Set | $\mathcal{O}^*(2^k)$ | $\mathcal{O}^*(3^k)$ | $\mathcal{O}^*(5^k)$ | ? |
| References | [9] | [15, 16] | [26] | here |

▶ **Theorem 1.2.** *Assuming SETH*[3]*, the following statements hold for all* $\varepsilon > 0$*:*

- *Connected Vertex Cover (with unit costs) cannot be solved in time* $\mathcal{O}^*((6-\varepsilon)^{\text{cw}})$*.*
- *Connected Dominating Set (with unit costs) cannot be solved in time* $\mathcal{O}^*((5-\varepsilon)^{\text{cw}})$*.*

This work is part of a research program to determine the optimal running times for connectivity problems relative to several width-parameters ranging from restrictive to more and more general ones, hence yielding a *fine-grained* understanding of the *price of generality*. We summarize the known results in Table 1. The cut-and-count-technique by Cygan et al. [15, 16] together with their lower bounds settle the complexity relative to treewidth (and pathwidth) for many connectivity problems. Bojikian et al. [9] consider the more restrictive *cutwidth* and combine cut-and-count with the rank-based approach to improve upon the treewidth-algorithms or provide new lower bound constructions with low cutwidth when no improved algorithm exists. Hegerfeld and Kratsch [26] consider the parameter *modular-treewidth* which lifts treewidth into the dense regime by combining tree decompositions with modular decompositions and thus serves as a natural intermediate step between treewidth and clique-width. The results on modular-treewidth are obtained by reducing directly to the treewidth-case or by applying the cut-and-count-technique and the modular structure to essentially reduce to a more involved problem parameterized by treewidth; in the latter case, new lower bound constructions are provided that follow similar high-level principles as here, but that have to adhere to different design restrictions. Cygan et al. [16] observe that connectivity increases the base by at most 1 in the sparse setting, e.g., Vertex Cover has optimal base 2, see Lokshtanov et al. [35], and Connected Vertex Cover has optimal base 3 parameterized by treewidth. For clique-width, this increase can be larger and the impact of connectivity can even flip the relative complexities, e.g., the optimal bases of Vertex Cover and Dominating Set are 2 and 4 [29, 32] which increase to 6 and 5, respectively, upon adding the connectivity constraint.

**Further Related Work.** Beyond these tight results, the cut-and-count-technique has also been applied to branchwidth [42] and treedepth [23, 38]. Due to its reliance on the isolation lemma, the cut-and-count-technique yields randomized algorithms. The rank-based approach of Bodlaender et al. [8] and the matroid-based techniques of Fomin et al. [19, 20] deal

[3] If we instead assume an appropriate random variant of SETH, then these reductions also rule out Monte-Carlo algorithms with such running times.

with this shortcoming at the cost of a higher running time and the rank-based approach can also be applied in other contexts. By combining the rank-based approach with other techniques to avoid Gaussian elimination, optimal running times can be obtained in some cases, such as for Hamiltonian Cycle parameterized by pathwidth [12, 13] or coloring problems parameterized by cutwidth [22, 31]. There are further applications of the rank-based approach to connectivity problems relative to dense width-parameters, such as rankwidth [5] and mim-width [3]. We also refer to the survey of Nederlof on rank-based methods [37].

Moving away from connectivity problems, we survey some more of the literature obtaining tight fine-grained parameterized algorithms for dense parameters. Iwata and Yoshida show that for any $\varepsilon > 0$ Vertex Cover can be solved in time $\mathcal{O}^*((2-\varepsilon)^{\mathrm{tw}})$ if and only if Vertex Cover can be solved in time $\mathcal{O}^*((2-\varepsilon)^{\mathrm{cw}})$ [29]; as the bases differ for treewidth and clique-width in our case, it seems difficult to transfer their techniques to our setting. Lampis [34] obtains the tight running time of $\mathcal{O}^*((2^q-2)^{\mathrm{cw}})$ for $q$-Coloring and a tight result for $q$-Coloring parameterized by a more restrictive variant of modular-treewidth. Generalizing to homomorphism problems, Ganian et al. [21] obtain tight results for parameterization by clique-width, where the obtained base depends on a special measure of the target graph. Katsikarelis et al. [32] obtain tight results for $(k,r)$-Center parameterized by clique-width and, in particular, the tight running time $\mathcal{O}^*(4^{\mathrm{cw}})$ for Dominating Set. Jacob et al. [30] and Hegerfeld and Kratsch [24] show that the running time $\mathcal{O}^*(4^{\mathrm{cw}})$ is tight for Odd Cycle Transversal, where the latter article also considers a generalization to more colors and contains tight results for parameters that do not fall into the class of width-parameters.

**Organization.** We discuss the preliminaries in Section 2. Section 3 covers the algorithms (Theorem 1.1) and Section 4 the lower bounds (Theorem 1.2); both sections first outline the used techniques and present more details for Connected Vertex Cover. For space reasons, we only give a few remarks regarding Connected Dominating Set. We conclude in Section 5. Proofs and sections that are delegated to the full version [25] are denoted by $\star$.

## 2 Preliminaries

For two integers $a, b$ we write $a \equiv_c b$ to indicate equality modulo $c \in \mathbb{N}$. We use Iverson's bracket notation: for a boolean predicate $p$, we have that $[p]$ is 1 if $p$ is true and 0 otherwise. For a function $f$ we denote by $f[v \mapsto \alpha]$ the function $(f \setminus \{(v, f(v))\}) \cup \{(v, \alpha)\}$, viewing $f$ as a set; we also write $f[v \mapsto \alpha, w \mapsto \beta]$ instead of $(f[v \mapsto \alpha])[w \mapsto \beta]$. By $\mathbb{Z}_2$ we denote the field of two elements. For $n_1, n_2 \in \mathbb{Z}$, we write $[n_1, n_2] = \{x \in \mathbb{Z} : n_1 \leq x \leq n_2\}$ and $[n_2] = [1, n_2]$. For a function $f\colon V \to \mathbb{Z}$ and a subset $W \subseteq V$, we write $f(W) = \sum_{v \in W} f(v)$. For functions $g\colon A \to B$, where $B \not\subseteq \mathbb{Z}$, and $A' \subseteq A$, we still denote the *image of $A'$ under $g$* by $g(A') = \{g(v) : v \in A'\}$. If $f\colon A \to B$ is a function and $A' \subseteq A$, then $f\big|_{A'}$ denotes the *restriction* of $f$ to $A'$ and for a subset $B' \subseteq B$, we denote the *preimage of $B'$ under $f$* by $f^{-1}(B') = \{a \in A : f(a) \in B'\}$. An ordered tuple of sets $(A_1, \ldots, A_\ell)$ is an *ordered subpartition* if $A_i \cap A_j = \emptyset$ for all $i \neq j \in [\ell]$. The *power set* of a set $A$ is denoted by $\mathcal{P}(A)$.

**Graph Notation.** We use common graph-theoretic notation and the essentials of parameterized complexity. Let $G = (V, E)$ be an undirected graph. For a vertex set $X \subseteq V$, we denote by $G[X]$ the subgraph of $G$ that is induced by $X$. The *open neighborhood* of a vertex $v$ is given by $N_G(v) = \{u \in V : \{u, v\} \in E\}$, whereas the *closed neighborhood* is given by $N_G[v] = N_G(v) \cup \{v\}$. For sets $X \subseteq V$ we define $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. For two disjoint vertex subsets $A, B \subseteq V$, we define $E_G(A, B) = \{\{a, b\} \in E(G) : a \in A, b \in B\}$

and adding a *join* between $A$ and $B$ means adding an edge between every vertex in $A$ and every vertex in $B$. For a vertex set $X \subseteq V$, we define $\delta_G(X) = E_G(X, V \setminus X)$ and we write $\delta_G(v) = \delta_G(\{v\})$ for single vertices $v$. We denote the *number of connected components* of $G$ by $\mathtt{cc}(G)$. A *cut* of $G$ is a partition $V = V_L \cup V_R$, $V_L \cap V_R = \emptyset$, of its vertices into two parts.

**Clique-Expressions and Clique-Width.** A *labeled graph* is a graph $G = (V, E)$ together with a *label function* $\mathtt{lab}\colon V \to \mathbb{N} = \{1, 2, 3, \ldots\}$; we usually omit mentioning $\mathtt{lab}$ explicitly. A labeled graph is *$k$-labeled* if $\mathtt{lab}(v) \leq k$ for all $v \in V$. We consider the following four operations on labeled graphs:

- the *introduce*-operation $\ell(v)$ constructs a single-vertex graph whose vertex $v$ has label $\ell$,
- the *union*-operation $G_1 \oplus G_2$ constructs the disjoint union of labeled graphs $G_1$ and $G_2$,
- the *relabel*-operation $\rho_{i \to j}(G)$ changes the label of all vertices in $G$ with label $i$ to label $j$,
- the *join*-operation $\eta_{i,j}(G)$, $i \neq j$, which adds an edge between every vertex in $G$ with label $i$ and every vertex in $G$ with label $j$.

A valid expression that only consists of introduce-, union-, relabel-, and join-operations is called a *clique-expression*. The graph constructed by a clique-expression $\mu$ is denoted $G_\mu$ and the constructed label function is denoted $\mathtt{lab}_\mu\colon V(G_\mu) \to \mathbb{N}$. We associate to a clique-expression $\mu$ the syntax tree $T_\mu$ in the natural way and to each node $t \in V(T_\mu)$ the corresponding operation. For any node $t \in V(T_\mu)$ the subtree rooted at $t$ induces a *subexpression* $\mu_t$. When $\mu$ is fixed, we define $G_t = G_{\mu_t}$, $V_t = V(G_t)$, $E_t = E(G_t)$, and $\mathtt{lab}_t = \mathtt{lab}_{\mu_t}$ for any $v \in V(T_\mu)$. We write $V_t^\ell = \mathtt{lab}_t^{-1}(\ell)$ for the set of all vertices with label $\ell$ at node $t$ and we write $L_t = \{\ell \in \mathbb{N} : V_t^\ell \neq \emptyset\}$ for the set of *nonempty labels at node $t$*.

A clique-expression $\mu$ is a *$k$-clique-expression* or just *$k$-expression* if $G_t$ is $k$-labeled for all $t \in V(T_\mu)$. The *clique-width* of a graph $G$, denoted by $\mathrm{cw}(G)$, is the minimum $k$ such that a $k$-expression $\mu$ with $G = G_\mu$ exists. A clique-expression $\mu$ is *linear* if in every union-operation the second graph consists only of a single vertex. Accordingly, we define the *linear-clique-width* of a graph $G$, denoted $\text{lin-cw}(G)$, by only considering linear expressions.

**Strong Exponential-Time Hypothesis.** The *Strong Exponential-Time Hypothesis* (SETH) [10, 28] concerns the complexity of $q$-SATISFIABILITY, i.e., every clause contains at most $q$ literals. We define $c_q = \inf\{\delta : q\text{-SATISFIABILITY can be solved in time } \mathcal{O}(2^{\delta n})\}$ for all $q \geq 3$. The weaker *Exponential-Time Hypothesis* (ETH) of Impagliazzo and Paturi [27] posits that $c_3 > 0$ and the Strong Exponential-Time Hypothesis states that $\lim_{q \to \infty} c_q = 1$. Equivalently, for every $\delta < 1$, there is some $q$ such that $q$-SATISFIABILITY cannot be solved in time $\mathcal{O}(2^{\delta n})$. For our lower bounds, the following weaker variant of SETH is sufficient.

▶ **Conjecture 2.1** (CNF-SETH). *For every $\varepsilon > 0$, there is no algorithm solving SATISFIABILITY with $n$ variables and $m$ clauses in time $\mathcal{O}(\mathrm{poly}(m)(2-\varepsilon)^n)$.*

**Cut and Count.** Let $G = (V, E)$ denote a connected graph. For easy reference, we repeat the key definition and lemmas of the cut-and-count-technique here.

▶ **Definition 2.2** ([16]). A cut $(V_L, V_R)$ of an undirected graph $G = (V, E)$ is *consistent* if $u \in V_L$ and $v \in V_R$ implies $\{u, v\} \notin E$, i.e., $E_G(V_L, V_R) = \emptyset$. A *consistently cut subgraph* of $G$ is a pair $(X, (X_L, X_R))$ such that $X \subseteq V$ and $(X_L, X_R)$ is a consistent cut of $G[X]$. We denote the set of consistently cut subgraphs of $G$ by $\mathcal{C}(G)$.

▶ **Lemma 2.3** ([16]). *Let $X$ be a subset of vertices such that $v_* \in X \subseteq V$. The number of consistently cut subgraphs $(X, (X_L, X_R))$ such that $v_* \in X_L$ is equal to $2^{\mathtt{cc}(G[X])-1}$.*

▶ **Corollary 2.4** (⋆). *Let $\mathcal{R} \subseteq \mathcal{P}(V)$ be a family of vertex sets so that every $X \in \mathcal{R}$ contains $v_*$. If the set $\mathcal{A} = \{(X, (X_L, X_R)) \in \mathcal{C}(G) : X \in \mathcal{R}, v_* \in X_L\}$ has odd cardinality, then there exists an $X \in \mathcal{R}$ such that $G[X]$ is connected.*

With the isolation lemma we avoid unwanted cancellations in the cut-and-count-technique.

▶ **Definition 2.5.** A function $\mathbf{w}\colon U \to \mathbb{Z}$ *isolates* a set family $\mathcal{F} \subseteq \mathcal{P}(U)$ if there is a unique $S' \in \mathcal{F}$ with $\mathbf{w}(S') = \min_{S \in \mathcal{F}} \mathbf{w}(S)$; recall that $\mathbf{w}(X) = \sum_{u \in X} \mathbf{w}(u)$ for subsets $X$ of $U$.

▶ **Lemma 2.6** (Isolation Lemma, [36]). *Let $\mathcal{F} \subseteq \mathcal{P}(U)$ be a nonempty set family over a universe $U$. Let $N \in \mathbb{N}$ and for each $u \in U$ choose a weight $\mathbf{w}(u) \in [N]$ uniformly and independently at random. Then $\mathbb{P}[\mathbf{w}$ isolates $\mathcal{F}] \geq 1 - |U|/N$.*

## 3 Dynamic Programming Algorithms

**Cut and Count.** The cut-and-count-technique by Cygan et al. [16] allows us to reduce the global connectivity constraint to a locally checkable counting problem. Consistent cuts, cf. Definition 2.2, are the main driver of the cut-and-count-technique. Any graph $G = (V, E)$ admits exactly $2^{\mathsf{cc}(G)}$ distinct consistent cuts, where $\mathsf{cc}(G)$ is the number of connected components of $G$. By fixing a vertex $v_*$ and only considering consistent cuts $(V_L, V_R)$ with $v_* \in V_L$, this number reduces to $2^{\mathsf{cc}(G)-1}$, so that $G$ admits an odd number of such consistent cuts if and only if $G$ is connected. This behavior implies that if we count pairs $(X, (X_L, X_R))$, where $v_* \in X \subseteq V$ is a partial solution and $(X_L, X_R)$ a consistent cut of $G[X]$ with $v_* \in X_L$, modulo two, that all disconnected solutions cancel. When multiple connected solutions exist, they could also cancel modulo two, but this issue can be avoided at the cost of randomization by using the isolation lemma, cf. Lemma 2.6. So, if there exists a connected solution, then we can assume that Lemma 2.3 applies and let the algorithm answer accordingly.

**Lifting Vertex States to Label States.** For dynamic programming along clique-expressions, we have to characterize the relevant interactions of a partial solution with the *labels* which govern which *joins* can be constructed by the expression. In the considered problems, a single vertex $v$ can take a constant number of different states with respect to a partial solution which we capture with a problem-dependent set $\Omega$; e.g., for Connected Vertex Cover, we have $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$, representing $v \notin X$ (state $\mathbf{0}$), $v \in X_L$ (state $\mathbf{1}_L$), and $v \in X_R$ (state $\mathbf{1}_R$), respectively. A clique-expression repeatedly adds *joins* between pairs of vertex sets, say $A$ and $B$, i.e., all possible edges between $A$ and $B$ are added, and the algorithm must check whether a partial solution remains feasible after adding a join and possibly update some states. A priori, each choice of vertex states in a label could yield different behaviors for partial solutions. However, for the considered problems the precise multiplicity of a vertex state in $A$ or in $B$ is irrelevant for a join; it suffices to distinguish for each side which vertex states appear and which do not. Therefore, the label states are captured by the subsets of $\Omega$. The next two techniques will allow us to reduce the number of considered states further.

**Nice Clique-Expressions.** We refine and augment standard clique-expressions to distinguish between *live* and *dead* labels. When performing dynamic programming along a clique-expression, we consider the induced subgraphs defined by subexpressions of the given clique-expression. At a subexpression, we say that a label $\ell$ is *live* if in the remaining expression the vertices with label $\ell$ receive further edges that are not present in the current subexpression, otherwise $\ell$ is *dead*. First, we observe that we do not need to track the states of a partial solution at the dead labels, as they only have trivial interactions with

the other states in the remaining expression. Hence, we only need to consider the states that can be attained at live labels which allows us to reduce the number of considered states for Connected Vertex Cover. To simplify the algorithms and avoid handling of edge cases, we transform the clique-expressions so that no degenerate cases occur and add a dead-operation $\bot_\ell$ which handles label $\ell$ turning from live to dead. The dead-operation is similar to *forget vertex nodes* in *nice tree decompositions* [33]. Distinguishing live and dead labels has been used before [21, 32, 34] to obtain improved running times, but handling the label types explicitly via an additional operation is new to the best of the authors' knowledge.

**Inclusion-Exclusion States.** For Connected Dominating Set, we transform to a different set of vertex states, called *inclusion-exclusion states*, which have proven helpful for domination problems before [23, 39, 41, 43]. These states do not track whether a vertex is *undominated* or *dominated* by a partial solution, but *allow* a vertex to be dominated or *forbid* it. A solution to the original problem can usually be recovered by an inclusion-exclusion argument, however when lifting to label states this argument does not directly transfer. We show that the argument can be adapted for label states when working modulo two, whereas for vertex states the argument is known to also work for non-modular counting. The advantage of the inclusion-exclusion states is that at joins we do not have to update vertex states from undominated to dominated, thus simplifying the algorithm and also allowing us to collapse several label states into a single one. The dead-operations of nice clique-expressions serve as suitable timepoints to apply the adapted inclusion-exclusion argument.

**Fast Convolutions.** To quickly compute the recurrences for union-operations, we utilize algorithms for *fast subset convolution*. We tailor the techniques developed by Björklund et al. [6] on trimmed subset convolutions to obtain a fast algorithm for the union-recurrence appearing in the Connected Vertex Cover algorithm. For Connected Dominating Set, we employ the lattice-based results of Björklund et al. [7].

## 3.1 Nice Clique-Expressions

Let $\mu$ be a $k$-expression for $G = (V, E)$; the associated syntax tree is $T_\mu$. We say that a clique-expression $\mu$ is *irredundant* if for any join-operation $\eta_{i,j}(G_{t'}) = t \in V(T_\mu)$, it holds that $E_{G_{t'}}(V^i_{t'}, V^j_{t'}) = \emptyset$, i.e., no edge added by the join existed before.

▶ **Theorem 3.1** ([11]). *Any $k$-expression $\mu$ can be transformed into an equivalent, i.e., $G_{\mu'} = G_\mu$, irredundant $k$-expression $\mu'$ in polynomial time.*

Irredundancy still allows several edge cases regarding empty labels to occur, which require special handling in the dynamic programming algorithms. To avoid this extra effort in the algorithms, we transform any clique-expression so that these edge cases do not occur.

▶ **Definition 3.2.** A clique-expression $\mu$ of $G$ is *nice* if $\mu$ satisfies the following properties:

- $\mu$ is irredundant,
- for every join-node $\eta_{i,j}(G_{t'}) = t \in V(T_\mu)$, where $t'$ is the child of $t$, we have that $G_t \neq G_{t'}$, i.e., $t$ adds at least one edge and $V^i_{t'} \neq \emptyset$ and $V^j_{t'} \neq \emptyset$,
- for every relabel-node $\rho_{i\to j}(G_{t'}) = t \in V(T_\mu)$, where $t'$ is the child of $t$, we have that $V^i_{t'} \neq \emptyset$ and $V^j_{t'} \neq \emptyset$.

In the full version of the paper, we give a short proof how to transform a $k$-expression into an equivalent nice $k$-expression. However, Ducoffe [17] has also shown how to perform such a transformation in only linear time with a more involved proof.

▶ **Lemma 3.3** (⋆, [17]). *Any $k$-expression $\mu$ can be transformed into an equivalent, i.e., $G_{\mu'} = G_\mu$, nice $k$-expression $\mu'$ in polynomial time.*

For nice clique-expressions, we will now present how we augment the associated syntax tree to distinguish between live and dead labels. For the remainder of this section, we assume that $G$ is a connected graph with at least two vertices. We begin with the following definition.

▶ **Definition 3.4.** Given a clique-expression $\mu$ for $G = (V, E)$ and a node $t \in V(T_\mu)$, the set of *dead vertices at* $t$ is defined by $D_t = \{v \in V_t : \delta_G(v) \subseteq E_t\}$. A vertex $v \in V_t \setminus D_t$ is called *live at* $t$.

The next lemma shows that in an irredundant clique-expression either none or all vertices in a label are dead, thus allowing us to sensibly speak of dead and live labels.

▶ **Lemma 3.5** (⋆). *Given an irredundant $k$-expression $\mu$ for $G = (V, E)$, a node $t \in V(T_\mu)$, and a nonempty label $\ell \in L_t$, we have that either $V_t^\ell \cap D_t = \emptyset$ or $V_t^\ell \subseteq D_t$.*

The following definition formalizes the handling of live and dead labels and the dead nodes that are added when a label turns from live to dead.

▶ **Definition 3.6.** Given an irredundant $k$-expression $\mu$ for $G = (V, E)$, the *augmented syntax tree* $\hat{T}_\mu$ of $\mu$ is obtained from $T_\mu$ by inserting up to two *dead nodes* directly above every join node $t = \eta_{i,j}(G_{t'})$, where $t'$ is the child of $t$ in $T_\mu$, based on the following criteria:

- if $V_t^i \subseteq D_t \setminus D_{t'}$, then the node $\bot_i$ is inserted,
- if $V_t^j \subseteq D_t \setminus D_{t'}$, then the node $\bot_j$ is inserted,
- if both nodes $\bot_i$ and $\bot_j$ are inserted, then we insert them in any order.

We extend the notations $G_t$, $V_t$, $D_t$, $V_t^\ell$, for $\ell \in [k]$, to dead nodes by inheriting the values of the child node. For every node $t \in V(\hat{T}_\mu)$, we inductively define the *live labels* $L_t^{live} \subseteq L_t$ by

$$
\begin{array}{llll}
L_t^{live} = \{\ell\} & \text{if } t = \ell(v), & L_t^{live} = L_{t'}^{live} \setminus \{i\} & \text{if } t = \rho_{i\to j}(G_{t'}),\\
L_t^{live} = L_{t'}^{live} & \text{if } t = \eta_{i,j}(G_{t'}), & L_t^{live} = L_{t'}^{live} \setminus \{\ell\} & \text{if } t = \bot_\ell(G_{t'}),\\
L_t^{live} = L_{t_1}^{live} \cup L_{t_2}^{live} & \text{if } t = G_{t_1} \oplus G_{t_2}. & &
\end{array}
$$

Dually, the set of *dead labels* $L_t^{dead} \subseteq L_t$ is given by $L_t^{dead} = L_t \setminus L_t^{live}$.

The next lemma shows that, up to pending dead nodes, $L_t^{live}$ contains all nonempty labels that only consist of live vertices at $t$. Due to Lemma 3.5, no label of an irredundant $k$-expression can contain both live and dead vertices simultaneously.

▶ **Lemma 3.7** (⋆). *Let $\mu$ be a nice $k$-expression of $G = (V, E)$ and $\hat{T}_\mu$ its augmented syntax tree. For any node $t \in V(\hat{T}_\mu)$ and $\ell \in L_t$, we have that $V_t^\ell \cap D_t = \emptyset$ implies $\ell \in L_t^{live}$. If $t$ is not the child of a dead node, then we even have that $V_t^\ell \cap D_t = \emptyset$ if and only if $\ell \in L_t^{live}$.*

## 3.2 Connected Vertex Cover

| Connected Vertex Cover | |
|---|---|
| **Input:** | An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$. |
| **Question:** | Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $G - X$ contains no edges and $G[X]$ is connected? |

Let $(G = (V, E), \mathbf{c}, \overline{b})$ be a Connected Vertex Cover instance with $\mathbf{c}(v) \leq |V|^{\mathcal{O}(1)}$ for all $v \in V$. Furthermore, we assume that $G$ is connected and contains at least two vertices.

Let $\mu$ be a nice $k$-expression for $G$. To apply the cut-and-count-technique, we first pick an edge in $G$, branch on one of its endpoints $v_*$, and in this branch only consider solutions containing $v_*$. Furthermore, we sample a weight function $\mathbf{w}\colon V \to [2|V|]$ for the isolation lemma, cf. Lemma 2.6. We perform bottom-up dynamic programming along the augmented syntax tree $\hat{T}_\mu$. At every node $t \in V(\hat{T}_\mu)$, we consider the following family of partial solutions

$$\mathcal{A}_t = \{(X, (X_L, X_R)) \in \mathcal{C}(G_t) : G_t - X \text{ contains no edges and } (v_* \in V_t \Rightarrow v_* \in X_L)\}.$$

In other words, $\mathcal{A}_t$ contains all consistently cut vertex covers of $G_t$ such that $v_*$ is on the left side of the cut if possible. For every $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, we define $\mathcal{A}_t^{\overline{c},\overline{w}} = \{(X, (X_L, X_R)) \in \mathcal{A}_t : \mathbf{c}(X) = \overline{c}, \mathbf{w}(X) = \overline{w}\}$. Let $\hat{r}$ denote the root node of the augmented syntax tree $\hat{T}_\mu$. By Corollary 2.4, there exists a connected vertex cover $X$ of $G$ with $\mathbf{c}(X) \leq \overline{b}$ if there exist $\overline{c} \in [0, \overline{b}]$ and $\overline{w} \in [0, \mathbf{w}(V)]$ such that $\mathcal{A}_{\hat{r}}^{\overline{c},\overline{w}}$ has odd cardinality.

We proceed by analyzing the behavior of a partial solution $(X, (X_L, X_R)) \in \mathcal{A}_t$ with respect to a label $V_t^\ell$, $\ell \in L_t$. A vertex $v \in V_t^\ell$ can take one of the states $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$, meaning respectively $v \notin X$, or $v \in X_L$, or $v \in X_R$. To check the feasibility of $(X, (X_L, X_R))$, it suffices to store for each label which vertex states appear and which do not, as the constraints implied by $\mathcal{A}_t$ are "CSP-like", i.e., they apply to every edge, and they can be evaluated for every join by considering all pairs of involved vertex states. Hence, the power set $\mathcal{P}(\Omega)$ of $\Omega$ captures all possible label states.

The power set $\mathcal{P}(\Omega)$ a priori yields eight different states per label. However, we can exclude the state $\emptyset$ and the state $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ from consideration. The former can be excluded, since we only need to store the state for nonempty labels, i.e., containing at least one vertex. The exclusion of the state $\Omega = \{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ is more subtle: any additional incident join would lead to an infeasible solution for this state, hence only dead labels, cf. Definition 3.6, may sensibly take this state. We return to this issue in a moment. Since it suffices to store the states of live labels, we set $\mathbf{States} = \mathcal{P}(\Omega) \setminus \{\emptyset, \Omega\} = \{\{\mathbf{0}\}, \{\mathbf{1}_L\}, \{\mathbf{1}_R\}, \{\mathbf{0}, \mathbf{1}_L\}, \{\mathbf{0}, \mathbf{1}_R\}, \{\mathbf{1}_L, \mathbf{1}_R\}\}$.

Given a node $t \in V(\hat{T}_\mu)$, a *$t$-signature* is a function $f\colon L_t^{live} \to \mathbf{States}$. For every node $t \in V(\hat{T}_\mu)$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$, and $t$-signature $f$, we define

$$\begin{aligned}\mathcal{A}_t^{\overline{c},\overline{w}}(f) = \{(X, (X_L, X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}} : \; & \mathbf{0} \in f(\ell) \Leftrightarrow V_t^\ell \not\subseteq X \text{ for all } \ell \in L_t^{live}, \\ & \mathbf{1}_L \in f(\ell) \Leftrightarrow X_L \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t^{live}, \\ & \mathbf{1}_R \in f(\ell) \Leftrightarrow X_R \cap V_t^\ell \neq \emptyset \text{ for all } \ell \in L_t^{live}\}.\end{aligned}$$

We claim that excluding the state $\Omega$ does not cause issues. Consider some node $t$ whose parent is not a dead node, and $(X, (X_L, X_R)) \in \mathcal{A}_t^{\overline{c},\overline{w}}$ such that there is a live label $\ell \in L_t^{live}$ for which the three cases $V_t^\ell \not\subseteq X$, $X_L \cap V_t^\ell \neq \emptyset$, and $X_R \cap V_t^\ell \neq \emptyset$ simultaneously occur. Since $\ell$ is live, there is some $v \in N_G(V_t^\ell) \setminus N_{G_t}(V_t^\ell)$ by Lemma 3.7. We claim that $(X, (X_L, X_R))$ cannot be extended to a consistently cut vertex cover $(X', (X'_L, X'_R))$ of $G' = G[V_t \cup \{v\}]$ (hence also not of $G$). If $v \notin X'$, then there is an uncovered edge in $G'$ between $V_t^\ell$ and $v$. If $v \in X'$, then some edge in $G'$ crosses the cut $(X'_L, X'_R)$ and so the cut cannot be consistent. Hence, partial solutions attaining the state $\{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ at a live label can be discarded.

Instead of computing the sets $\mathcal{A}_t^{\overline{c},\overline{w}}(f)$ directly, we only compute the quantities $A_t^{\overline{c},\overline{w}}(f) = |\mathcal{A}_t^{\overline{c},\overline{w}}(f)| \mod 2$. The recurrences for computing $A_t^{\overline{c},\overline{w}}(f)$, for every $t \in V(\hat{T}_\mu)$, $t$-signature $f$, $\overline{c} \in [0, \mathbf{c}(V)]$, $\overline{w} \in [0, \mathbf{w}(V)]$ depend on the type of the considered node $t$:

**Introduce node.** If $t = \ell(v)$ for some $\ell \in [k]$, then $L_t^{live} = \{\ell\}$ and

$$\begin{aligned}A_t^{\overline{c},\overline{w}}(f) = {} & [v \neq v_* \vee f(\ell) = \{\mathbf{1}_L\}] \\ & \cdot [(f(\ell) = \{\mathbf{0}\} \wedge \overline{c} = \overline{w} = 0) \vee (f(\ell) \in \{\{\mathbf{1}_L\}, \{\mathbf{1}_R\}\} \wedge \overline{c} = \mathbf{c}(v) \wedge \overline{w} = \mathbf{w}(v))],\end{aligned}$$

since in a singleton graph any choice of singleton state leads to a valid solution, but if $v = v_*$ then only the solution with $v_*$ on the left side of the cut is allowed.

**Relabel node.** If $t = \rho_{i\to j}(G_{t'})$, where $t'$ is the child of $t$, for some $i, j \in [k]$, then by niceness of $\mu$ it follows that $i \in L_{t'}$, $j \in L_{t'}$, $L_t = L_{t'} \setminus \{i\}$ and either $i, j \in L_{t'}^{live}$ or $i, j \in L_{t'}^{dead}$.

- If labels $i$ and $j$ are live at $t'$, then label $j$ is live at $t$ and the recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\substack{\mathbf{S}_1,\mathbf{S}_2\in\mathbf{States}:\\ \mathbf{S}_1\cup\mathbf{S}_2=f(j)}} A_{t'}^{\overline{c},\overline{w}}(f[i\mapsto \mathbf{S}_1, j\mapsto \mathbf{S}_2]),$$

  since $V_t^j = V_{t'}^i \cup V_{t'}^j$ and we simply have to iterate over all possible combinations of previous states at labels $i$ and $j$ that yield the desired state $f(j)$.
- If labels $i$ and $j$ are dead at $t'$, then label $j$ is dead at $t$ and since we do not track the state of dead labels, we can simply copy the previous table, i.e., $A_t^{\overline{c},\overline{w}}(f) = A_{t'}^{\overline{c},\overline{w}}(f)$.

**Join node.** To check whether two states can lead to a feasible solution after adding a join between their labels, we introduce a helper function feas: $\mathbf{States} \times \mathbf{States} \to \{0,1\}$ defined by $\text{feas}(\mathbf{S}_1, \mathbf{S}_2) = [\mathbf{0} \notin \mathbf{S}_1 \vee \mathbf{0} \notin \mathbf{S}_2][\mathbf{1}_L \in \mathbf{S}_1 \Rightarrow \mathbf{1}_R \notin \mathbf{S}_2][\mathbf{1}_R \in \mathbf{S}_1 \Rightarrow \mathbf{1}_L \notin \mathbf{S}_2]$. There are two reasons for infeasibility: a join edge is not covered, i.e., $\mathbf{0}$ appears on both sides, or a join edge connects both sides of the cut, i.e., $\mathbf{1}_L$ appears on one side and $\mathbf{1}_R$ on the other.

We have $t = \eta_{i,j}(G_{t'})$ for some $i \neq j \in L_{t'}$, where $t'$ is the child of $t$. We have $i, j \in L_{t'}^{live}$ and if vertices turn dead at $t$, i.e., $D_{t'} \subsetneq D_t$, then a future dead node will handle it. Hence, we simply filter out all partial solutions that become infeasible due to the new join:

$$A_t^{\overline{c},\overline{w}}(f) = \text{feas}(f(i), f(j)) A_{t'}^{\overline{c},\overline{w}}(f).$$

**Dead node.** We have that $t = \bot_\ell(G_{t'})$, where $t'$ is the child of $t$, $\ell \notin L_t^{live}$, and $L_t^{live} = L_{t'}^{live} \setminus \{\ell\}$. Since the only change is that $t$-signatures do not track the state of label $\ell$ anymore, we add up the contributions of all previous states of label $\ell$. Hence, the recurrence is given by

$$A_t^{\overline{c},\overline{w}}(f) = \sum_{\mathbf{S}\in\mathbf{States}} A_{t'}^{\overline{c},\overline{w}}(f[\ell\mapsto \mathbf{S}]).$$

Although this recurrence looks simple, its correctness proof is nontrivial as it relies on the previous argument why label state $\Omega$ can be excluded.

**Union node.** We omit the standard, but somewhat technical, description of the union-recurrence here. After handling labels that are nonempty at only one of the children, a trimmed subset convolution remains that we can solve in time $\mathcal{O}^*(6^{|L_t^{live}|})$ for all $\overline{c}$, $\overline{w}$, and $f$ simultaneously via the convolution algorithms developed in the appendix of the full version.

▶ **Lemma 3.8** (⋆). *Given a nice $k$-expression $\mu$ of $G = (V, E)$, the quantities $A_t^{\overline{c},\overline{w}}(f)$ for all nodes $t \in V(\hat{T}_\mu)$, $t$-signatures $f$, and appropriate $\overline{c}$, $\overline{w}$, can be computed in time $\mathcal{O}^*(6^k)$.*

**Proof sketch.** For introduce nodes, relabel nodes, or join nodes, the recurrence for $A_t^{\overline{c},\overline{w}}(f)$ can be computed in polynomial time, as additions and multiplications in $\mathbb{Z}_2$ take constant time. For union nodes $t$, we compute the recurrence for all $f$, $\overline{c}$, $\overline{w}$ simultaneously in time $\mathcal{O}^*(6^{|L_t^{live}|})$ as discussed. As $\mu$ is a $k$-expression, we have $|L_t^{live}| \leq k$ for all $t \in V(\hat{T}_\mu)$ and in particular at most $6^k$ $t$-signatures for any node $t \in V(T_\mu)$. Hence, the running time follows.

The proof of correctness for introduce nodes, relabel nodes, join nodes, and union nodes is straightforward. For dead nodes, the proof of correctness follows from the discussion on the exclusion of state $\{\mathbf{0}, \mathbf{1}_L, \mathbf{1}_R\}$ and the construction of the augmented syntax tree $\hat{T}_\mu$. ◀

▶ **Theorem 3.9** (⋆). *There is a randomized algorithm that given a nice $k$-expression $\mu$ for a graph $G = (V, E)$ can solve* Connected Vertex Cover *in time $\mathcal{O}^*(6^k)$. The algorithm does not return false positives and returns false negatives with probability at most $1/2$.*

**Proof.** We sample a weight function $\mathbf{w}\colon V \to [2n]$ uniformly at random. Then, we pick an edge in $G$ and branch on its endpoints; the chosen endpoint takes the role of $v_*$ in the current branch. Using Lemma 3.8, we compute the quantities $A_t^{\overline{c},\overline{w}}(f)$. At the root node $\hat{r}$, we have that $L_{\hat{r}}^{live} = \emptyset$. The algorithm returns true if in one of the branches there is some choice of $\overline{c} \leq \overline{b}$, $\overline{w} \in [0, 2n^2]$, such that $A_{\hat{r}}^{\overline{c},\overline{w}}(\emptyset) \neq 0$, otherwise the algorithm returns false.

The running time directly follows from Lemma 3.8. The correctness and error probability follow from Corollary 2.4, Lemma 2.3 and the isolation lemma, Lemma 2.6. Since these arguments are standard for the cut-and-count-technique, we omit them here. ◀

## 3.3 Remarks on Connected Dominating Set Algorithm (⋆)

| Connected Dominating Set | |
|---|---|
| **Input:** | An undirected graph $G = (V, E)$, a cost function $\mathbf{c}\colon V \to \mathbb{N} \setminus \{0\}$ and an integer $\overline{b}$. |
| **Question:** | Is there a set $X \subseteq V$, $\mathbf{c}(X) \leq \overline{b}$, such that $N(X) \cup X = V$ and $G[X]$ is connected? |

To obtain our algorithm for Connected Dominating Set, we transform to the inclusion-exclusion states and apply the cut-and-count-technique. We again have the vertex states $\mathbf{1}_L$ and $\mathbf{1}_R$, but the state $\mathbf{0}$ splits into the *allowed* ($\mathbf{A}$) and *forbidden* state ($\mathbf{F}$), which denote that a vertex is allowed or forbidden to be dominated. Lifting to label states, we see that the presence of allowed vertices is irrelevant, as they impose no constraint on future joins, and that all label states containing at least two distinct non-allowed states behave in the same way. This allows us to collapse the number of considered label states down to five.

To count solutions dominating a vertex $v$ with the inclusion-exclusion states, one usually subtracts the number of solutions where $v$ is forbidden from the solutions where $v$ is allowed. This argument fails when applied to labels, i.e., *groups* of vertices. Instead, our dynamic program counts solutions with a label containing $u$ undominated vertices exactly $2^u$ times, so that all solutions with $u > 0$ cancel modulo two. Whenever a label turns dead, we apply this argument to ensure that all vertices in dead labels are dominated.

# 4 Lower Bounds

**Construction Principle.** The high-level construction principle of the lower bounds follows the style of Lokshtanov et al. [35]. That means the resulting graphs can be interpreted as a *matrix of blocks*, where each block spans several rows and columns. Every row is a long *path-like gadget* that simulates a constant number of variables of the Satisfiability instance and which contributes 1 unit of clique-width. The number of simulated variables is tied to the running time that we want to rule out. For technical reasons, we consider bundles of rows simulating a *variable group* of appropriate size. Every column corresponds to a clause and consists of gadgets that *decode* the states on the path gadgets and check whether the resulting assignment satisfies the clause. As a consequence of the construction principle, the lower bounds already apply to *linear* clique-width.

**Path Gadgets and State Transitions.** Our main contribution is the design of the *path gadgets* that lie at the intersection of every row and column, whereas the *decoding gadgets* can be adapted from Cygan et al. [15]. To ensure that each row contributes one unit of

clique-width, adjacent path gadgets in a row are connected by a join. Our goal is to design a path gadget admitting as many states as possible. An important issue is how the state of the path gadgets may transition along each row, as the reduction only works when the state transitions follow some *transition order*, i.e, state $i$ can transition to state $j$ only if $i \leq j$.

**Determining the Transition Order.** For sparse width-parameters such as pathwidth, determining an appropriate transition order is much simpler, as the number of possible states is very small, e.g., there are at most four vertex states for the considered benchmark problems. The possible set of states for clique-width is much larger and we must select a specific subset of states, as not all of them admit a transition order. Hence, we analyze the possible state transitions across a join, obtaining a *transition/compatibility* matrix showing which pairs of states can lead to a globally feasible solution and which cannot. After possibly reordering the rows and columns of the compatibility matrix, a transition order must induce a *triangular submatrix* with ones on the diagonal. From a largest possible such triangular submatrix of the compatibility matrix, we can then deduce an appropriate transition order which guides the design of the path gadget.

**Anatomy of a Path Gadget.** Our path gadgets consist of a *central clique* communicating with the decoding gadgets, and two *boundary* parts, i.e., the *left* and *right* part connecting to the previous and following join. In the central clique, each solution avoids exactly one vertex representing the state of the path gadget. To implement the transition order, the left and right part have to communicate appropriate states to the two adjacent path gadgets. By *pairing* states along the main diagonal of the triangular submatrix, we see which states must be communicated in each case. The central idea behind designing the left and right part is to isolate the constituent state properties of the boundary vertices, such as, whether they are contained in the partial solution or whether they are dominated. This simplifies the communication with the central clique and expedites implementing the transition order.

**Root-Connectivity.** To capture the connectivity constraint, we create a distinguished vertex $\hat{r}$ called the *root* and attach a vertex of degree 1 to ensure that every connected vertex cover or connected dominating set must contain $\hat{r}$. Given a vertex subset $X \subseteq V(G)$ with $\hat{r} \in X$, we say that a vertex $v \in X$ is *root-connected* in $X$ if there is a $v, \hat{r}$-path in $G[X]$. We will just say *root-connected* if $X$ is clear from the context. The graph $G[X]$ is connected if and only if all vertices of $X$ are root-connected in $X$. For the state of a partial solution $X$, it is important to consider which vertices are root-connected in $X$ and which are not.

## 4.1 Path Gadget for Connected Vertex Cover

This subsection is devoted to constructing and analyzing the path gadget used to prove that Connected Vertex Cover (with unit costs) cannot be solved in time $\mathcal{O}^*((6-\varepsilon)^{\text{lin-cw}(G)})$ for some $\varepsilon > 0$ unless the CNF-SETH fails. The remaining parts of the construction are standard and can be found in the full version. We build a path gadget admitting 6 distinct states which narrows down to a single label/join, so that each row contributes one unit of linear clique-width. Each single vertex $v$ has one of 3 states with respect to a partial solution $X$: $v \notin X$ (state $\mathbf{0}$), $v \in X$ and $v$ is root-connected (state $\mathbf{1}_1$) or not root-connected (state $\mathbf{1}_0$). The state of a label can be described as a subset of $\{\mathbf{0}, \mathbf{1}_0, \mathbf{1}_1\}$.

We proceed by studying the compatibility of theses label states across a join, but we will only give an informal description here. Essentially, we assume that the considered join is the final opportunity for two partial solutions $X_1, X_2 \subseteq V$ with $\hat{r} \in X_i$, $i \in [2]$, living on

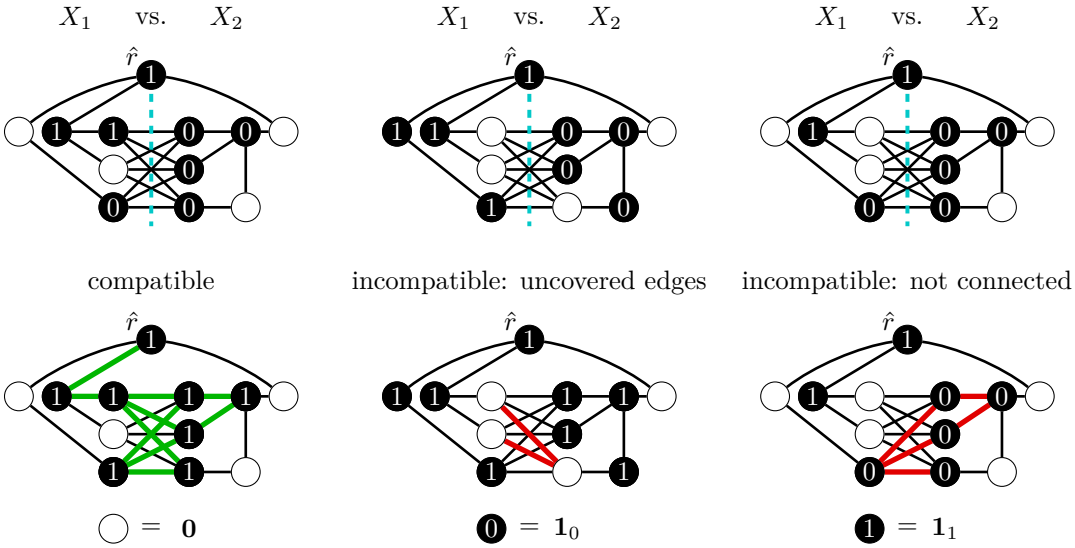



**Figure 1** Several cases of partial solution compatibility across a join. The first row depicts the vertex states in $X_1$ and $X_2$, separated by the dashed line. The second row depicts the vertex states in $X_1 \cup X_2$ and highlights, from left to right, the induced edges, the uncovered edges, and a connected component not containing the root $\hat{r}$.

**Table 2** A large triangular submatrix in the compatibility matrix of Connected Vertex Cover. The rows and columns have been reordered.

| $X_1$ vs. $X_2$ | $\{\mathbf{0}\}$ | $\{\mathbf{1}_0, \mathbf{0}\}$ | $\{\mathbf{1}_0\}$ | $\{\mathbf{1}_1, \mathbf{0}\}$ | $\{\mathbf{1}_1, \mathbf{1}_0\}$ | $\{\mathbf{1}_1\}$ |
|---|---|---|---|---|---|---|
| $\{\mathbf{1}_1\}$ | 1 | 1 | 1 | 1 | 1 | 1 |
| $\{\mathbf{1}_1, \mathbf{1}_0\}$ | 0 | 1 | 1 | 1 | 1 | 1 |
| $\{\mathbf{1}_1, \mathbf{0}\}$ | 0 | 0 | 1 | 0 | 1 | 1 |
| $\{\mathbf{1}_0\}$ | 0 | 0 | 0 | 1 | 1 | 1 |
| $\{\mathbf{1}_0, \mathbf{0}\}$ | 0 | 0 | 0 | 0 | 1 | 1 |
| $\{\mathbf{0}\}$ | 0 | 0 | 0 | 0 | 0 | 1 |

separate sides of the join (with the exception of $\hat{r}$) to connect. Hence, the partial solutions $X_1$ and $X_2$ are considered to be *compatible* when in $X_1 \cup X_2$ every vertex incident to the considered join has state $\mathbf{0}$ or $\mathbf{1}_1$ and every edge of the join is covered by $X_1 \cup X_2$; see Figure 1. Since the interaction of $X_i$, $i \in [2]$, with the respective side of the join is captured by the aforementioned states, we obtain a compatibility matrix of size $7 \times 7$.

In this compatibility matrix, we find the triangular submatrix depicted in Table 2, after reordering rows and columns. Independent sets of size two are sufficient to generate the relevant label states and they are represented by the following ordered pairs of vertex states: $(\mathbf{0}, \mathbf{0})$, $(\mathbf{1}_0, \mathbf{0})$, $(\mathbf{1}_0, \mathbf{1}_0)$, $(\mathbf{1}_1, \mathbf{0})$, $(\mathbf{1}_1, \mathbf{1}_0)$, $(\mathbf{1}_1, \mathbf{1}_1)$. Pairing these states along the diagonal, we learn which states should be communicated to the left and right boundary in each case.

**States.** We define the three atomic states $\mathbf{Atoms} = \{\mathbf{0}, \mathbf{1}_0, \mathbf{1}_1\}$, with their usual interpretation, and two predicates $\mathtt{sol}, \mathtt{conn}\colon \mathbf{Atoms} \to \{0, 1\}$ by $\mathtt{sol}(\mathbf{a}) = [\mathbf{a} \in \{\mathbf{1}_0, \mathbf{1}_1\}]$ and $\mathtt{conn}(\mathbf{a}) = [\mathbf{a} = \mathbf{1}_1]$. We define six (gadget) states consisting of four atomic states each:

$$\mathbf{s}^1 = (\mathbf{0}\ , \mathbf{0}\ , \mathbf{1}_1, \mathbf{1}_1), \quad \mathbf{s}^2 = (\mathbf{1}_0, \mathbf{0}\ , \mathbf{1}_1, \mathbf{1}_0), \quad \mathbf{s}^3 = (\mathbf{1}_0, \mathbf{1}_0, \mathbf{1}_1, \mathbf{0}\ ),$$
$$\mathbf{s}^4 = (\mathbf{1}_1, \mathbf{0}\ , \mathbf{1}_0, \mathbf{1}_0), \quad \mathbf{s}^5 = (\mathbf{1}_1, \mathbf{1}_0, \mathbf{1}_0, \mathbf{0}\ ), \quad \mathbf{s}^6 = (\mathbf{1}_1, \mathbf{1}_1, \mathbf{0}\ , \mathbf{0}\ ).$$

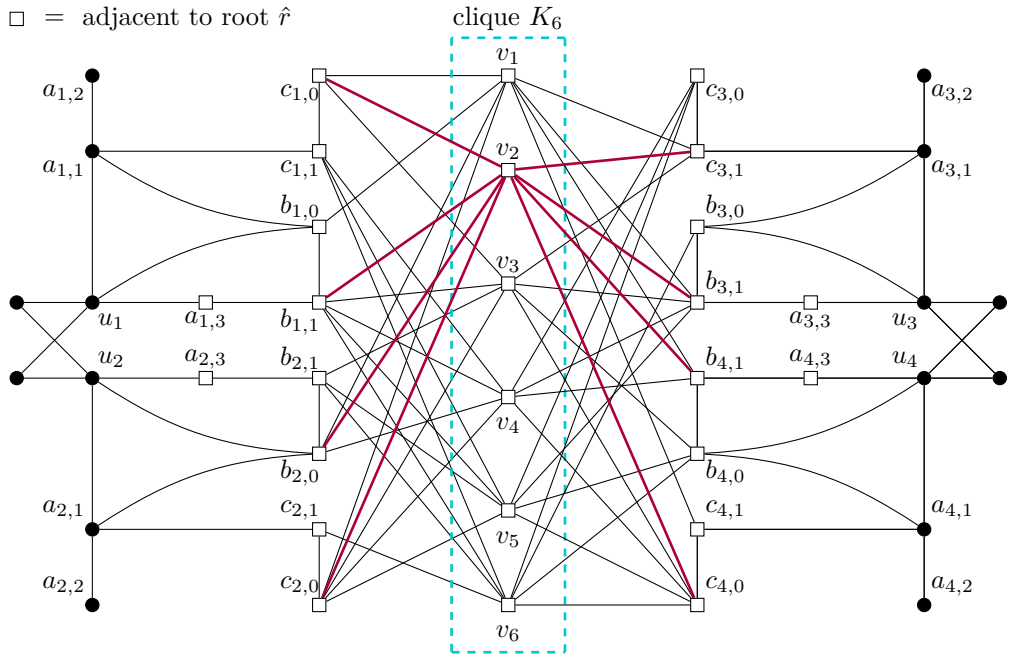



**Figure 2** The path gadget $P$ with the join vertices $u_1, u_2$ and $u_3, u_4$ joined to further vertices. All vertices depicted by a rectangle are adjacent to the root $\hat{r}$. The vertices inside the cyan dashed rectangle induce a clique. We have highlighted the edges incident to $v_2$ as an example.

The gadget states are numbered in the transition order. We collect the six gadget states in the set $\mathbf{States} = \{\mathbf{s}^1, \ldots, \mathbf{s}^6\}$ and use the notation $\mathbf{s}^\ell_i \in \mathbf{Atoms}$, $i \in [4]$, $\ell \in [6]$, to refer to the $i$-th atomic component of state $\mathbf{s}^\ell$. Given a partial solution $Y \subseteq V(G)$, we associate to each vertex its state in $Y$ with the map $\mathbf{state}_Y \colon V(G) \setminus \{\hat{r}\} \to \mathbf{Atoms}$, which is defined by $\mathbf{state}_Y(v) = \mathbf{0}$ if $v \notin Y$; $\mathbf{state}_Y(v) = \mathbf{1}_0$ if $v \in Y$ and $v$ is not root-connected in $Y \cup \{\hat{r}\}$; $\mathbf{state}_Y(v) = \mathbf{1}_1$ if $v \in Y$ and $v$ is root-connected in $Y \cup \{\hat{r}\}$.

**Construction.** The construction of the path gadget $P$ is as follows. We create 4 *join* vertices $u_1, \ldots, u_4$, 12 *auxiliary* vertices $a_{1,1}, a_{1,2}, a_{1,3}, a_{2,1}, \ldots, a_{4,3}$, 8 *solution indicator* vertices $b_{1,0}, b_{1,1}, b_{2,0}, b_{2,1}, \ldots, b_{4,1}$, 8 *connectivity indicator* vertices $c_{1,0}, c_{1,1}, c_{2,0}, c_{2,1}, \ldots, c_{4,1}$ and 6 *clique* vertices $v_1, \ldots, v_6$. We add edges so that the clique vertices $v_\ell$, $\ell \in [6]$, induce a clique of size 6. Next, we explain how to connect the indicator vertices to the clique vertices. The clique vertex $v_\ell$ corresponds to choosing state $\mathbf{s}^\ell$ on the join vertices $(u_1, u_2, u_3, u_4)$. The desired behavior of $P$ is that a partial solution $X$ of $P + \hat{r}$ contains $b_{i,1}$ if and only if $X$ contains $u_i$ and for the connectivity indicators, that $X$ contains $c_{i,1}$ if and only if $X$ contains $u_i$ and $u_i$ is root-connected in $X$. Accordingly, for all $i \in [4]$ and $\ell \in [6]$, we add the edges $\{v_\ell, b_{i,\mathtt{sol}(\mathbf{s}^\ell_i)}\}$ and $\{v_\ell, c_{i,\mathtt{conn}(\mathbf{s}^\ell_i)}\}$. For the remaining edges, we refer to Figure 2.

**Behavior of a Single Path Gadget.** We assume that $G$ is a graph that contains $P + \hat{r}$ as an induced subgraph and that only the join vertices $u_i, i \in [4]$, and clique vertices $v_\ell, \ell \in [6]$, have neighbors outside this copy of $P + \hat{r}$. Furthermore, let $X$ be a connected vertex cover of $G$ with $\hat{r} \in X$; we abuse notation and write $X \cap P$ instead of $X \cap V(P)$.

We begin by showing a lower bound for $|X \cap P|$ via a vertex-disjoint packing of subgraphs.

▶ **Lemma 4.1** (⋆). *We have that* $|X \cap P| \geq 21 = 4 \cdot 4 + 5$ *and more specifically* $a_{i,1} \in X$, $|X \cap \{u_i, a_{i,3}, b_{i,1}, b_{i,0}\}| \geq 2$, $|X \cap \{c_{i,0}, c_{i,1}\}| \geq 1$ *for all* $i \in [4]$ *and* $|X \cap \{v_1, \ldots, v_6\}| \geq 5$.

Using Lemma 4.1, we precisely analyze the solutions that match the lower bound of 21 on $P$ and show that these have the desired state behavior. We define for any $Y \subseteq V(P)$ the 4-tuple $\mathbf{state}(Y) = (\mathbf{state}_Y(u_1), \mathbf{state}_Y(u_2), \mathbf{state}_Y(u_3), \mathbf{state}_Y(u_4))$ and show that the states communicated to the boundary depend on the state of the central clique as desired.

▶ **Lemma 4.2** (⋆). *If* $|X \cap P| \leq 21$, *then* $|X \cap P| = 21$ *and* $a_{i,1} \in X$, $X \cap \{u_i, a_{i,3}, b_{i,1}, b_{i,0}\} \in \{\{u_i, b_{i,1}\}, \{a_{i,3}, b_{i,0}\}\}$, $|X \cap \{c_{i,0}, c_{i,1}\}| = 1$ *for all* $i \in [4]$ *and* $|X \cap \{v_1, \ldots, v_6\}| = 5$. *Furthermore, we have* $\mathbf{state}(X \cap P) = \mathbf{s}^\ell$ *for the unique integer* $\ell \in [6]$ *with* $v_\ell \notin X$.

**Proof sketch.** The first part follows by observing that the inequalities of Lemma 4.1 must be tight and that we must pick an antipodal pair in the cycle $\{u_i, a_{i,3}, b_{i,1}, b_{i,0}\}$. For the second part, we show that $\mathbf{state}_{X \cap P}(u_i) = \mathbf{s}_i^\ell$ for all $i \in [4]$. If $v_\ell \notin X$, then $b_{i,\mathtt{sol}(\mathbf{s}_i^\ell)}, c_{i,\mathtt{conn}(\mathbf{s}_i^\ell)} \in X$ by construction of $P$ and because $X$ is a vertex cover. Using the packing equations, we can propagate this information to $u_i$ and obtain the desired state. ◀

Similarly, we also establish that for every state $\mathbf{s}^\ell \in \mathbf{States}$ a partial solution $X_P^\ell$ attaining $\mathbf{s}^\ell$ actually exists. This is made precise by the following Lemma 4.3, which also shows that for these partial solutions it is sufficient to establish root-connectivity for the join vertices.

▶ **Lemma 4.3.** *For every* $\ell \in [6]$, *there exists a vertex cover* $X_P^\ell$ *of* $P$ *such that* $|X_P^\ell| = 21$, $X_P^\ell \cap \{v_1, \ldots, v_6\} = \{v_1, \ldots, v_6\} \setminus \{v_\ell\}$, *and* $\mathbf{state}(X_P^\ell) = \mathbf{s}^\ell$. *If* $X$ *is a vertex cover of* $G$ *with* $\hat{r} \in X$ *and* $X \cap P = X_P^\ell$ *and for every* $i \in [4]$ *either* $u_i \notin X$ *or* $u_i$ *is root-connected in* $X$, *then every vertex of* $X_P^\ell$ *is root-connected in* $X$.

**State Transitions.** To study the state transitions, suppose that we have two copies $P^1$ and $P^2$ of $P$ such that the vertices $u_3$ and $u_4$ in $P^1$ are joined to the vertices $u_1$ and $u_2$ in $P^2$. We denote the vertices of $P^1$ with a superscript 1 and the vertices of $P^2$ with a superscript 2, e.g., $u_3^1$ refers to the vertex $u_3$ of $P^1$. Again, suppose that $P^1$ and $P^2$ are embedded as induced subgraphs in a larger graph $G$ with a root vertex $\hat{r}$ and that only the vertices $u_1^1, u_2^1, u_3^2, u_4^2$ and the clique vertices $v_\ell^1, v_\ell^2$, $\ell \in [6]$, have neighbors outside of $P^1 + P^2 + \hat{r}$.

Using the previous lemmas, we show that the state transitions respect the transition order and that it is also feasible for the state to remain stable.

▶ **Lemma 4.4** (⋆). *Suppose that* $|X \cap P^1| \leq 21$ *and* $|X \cap P^2| \leq 21$, *then* $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ *and* $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$ *with* $\ell_1 \leq \ell_2$. *Additionally, for each* $\ell \in [6]$, *the set* $X^\ell = X_{P^1}^\ell \cup X_{P^2}^\ell$ *is a vertex cover of* $P^1 + P^2$ *with* $\mathbf{state}_{X^\ell}(\{u_3^1, u_4^1, u_1^2, u_2^2\}) \subseteq \{\mathbf{0}, \mathbf{1}_1\}$.

**Proof sketch.** By Lemma 4.2, we have $\mathbf{state}(X \cap P^1) = \mathbf{s}^{\ell_1}$ and $\mathbf{state}(X \cap P^2) = \mathbf{s}^{\ell_2}$ for some $\ell_1, \ell_2 \in [6]$. Showing $\ell_1 \leq \ell_2$ corresponds to proving that all entries below the main diagonal of the chosen submatrix of the compatibility matrix are zero, i.e., that the submatrix is *triangular*. The second part corresponds to checking that all diagonal entries are ones. Both parts are proved by case checking. ◀

## 4.2 Remarks on Connected Dominating Set Lower Bound (⋆)

The path gadget construction for Connected Dominating Set is very similar to Connected Vertex Cover as large parts of the gadget can be reused by subdividing edges. For Connected Dominating Set, we have to work with 4 vertex states instead of 3 due to tracking whether a vertex is dominated or not. Hence, we have to contend with, a priori,

$15 = 2^4 - 1$ possible label states instead of $7 = 2^3 - 1$. Surprisingly however, there are only five different behaviors for these label states, so that we obtain a smaller base, namely 5, for Connected Dominating Set compared to Connected Vertex Cover.

## 5 Conclusion and Open Problems

We have provided the first tight results under SETH for connectivity problems parameterized by clique-width, namely the problems Connected Vertex Cover and Connected Dominating Set. For several important benchmark problems such as Steiner Tree, Connected Odd Cycle Transversal, and Feedback Vertex Set, we are not able to achieve tight results with the current techniques. For Steiner Tree, our algorithmic techniques readily yield an $\mathcal{O}^*(4^{\mathrm{cw}})$-time algorithm, but the compatibility matrix for the lower bound only contains a triangular submatrix of size $3 \times 3$, hence we are not able to prove a larger lower bound than for treewidth. Similarly for Connected Odd Cycle Transversal, the techniques for Connected Vertex Cover yield an $\mathcal{O}^*(14^{\mathrm{cw}})$-time algorithm and a larger lower bound can be proven by adapting the gadgets for Connected Vertex Cover and adding a gadget to detect the used color at join-vertices, however there is again no large enough triangular submatrix that would allow us to show that $\mathcal{O}^*(14^{\mathrm{cw}})$ is optimal. For Feedback Vertex Set, a problem with a negative connectivity constraint in the form of acyclicity, the usual cut-and-count approach involves counting the edges induced by a partial solution, but this immediately leads to an XP-algorithm parameterized by clique-width as already noted by Bergougnoux and Kanté [4, 5]. Hence, a different approach is required to obtain plausible running times for tight results.

A big caveat in applying algorithms parameterized by clique-width is that we are lacking good algorithms for computing clique-expressions. The currently best algorithms rely on approximating clique-width via rankwidth, see Oum and Seymour [40] for the first such algorithm and Fomin and Korhonen [18] for the most recent one. However, the approximation via rankwidth introduces an exponential error, therefore all single-exponential algorithms parameterized by clique-width become double-exponential algorithms unless we are given a clique-expression by other means. A first step towards better approximation algorithms for clique-width could be a fixed-parameter tractable algorithm with subexponential error.

## References


1 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

2 Benjamin Bergougnoux. *Matrix decompositions and algorithmic applications to (hyper)graphs*. PhD thesis, University of Clermont Auvergne, Clermont-Ferrand, France, 2019. URL: https://tel.archives-ouvertes.fr/tel-02388683.

3 Benjamin Bergougnoux, Jan Dreier, and Lars Jaffke. *A logic-based algorithmic meta-theorem for mim-width*, pages 3282–3304. SIAM, 2023. doi:10.1137/1.9781611977554.ch125.

4 Benjamin Bergougnoux and Mamadou Moustapha Kanté. Fast exact algorithms for some connectivity problems parameterized by clique-width. *Theor. Comput. Sci.*, 782:30–53, 2019. doi:10.1016/j.tcs.2019.02.030.

5 Benjamin Bergougnoux and Mamadou Moustapha Kanté. More applications of the d-neighbor equivalence: Acyclicity and connectivity constraints. *SIAM J. Discret. Math.*, 35(3):1881–1926, 2021. doi:10.1137/20M1350571.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Trimmed moebius inversion and graphs of bounded degree. *Theory Comput. Syst.*, 47(3):637–654, 2010. doi:10.1007/s00224-009-9185-7.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast zeta transforms for lattices with few irreducibles. *ACM Trans. Algorithms*, 12(1):4:1–4:19, 2016. doi:10.1145/2629429.

8 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.

9 Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.14.

10 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In Jianer Chen and Fedor V. Fomin, editors, *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, 2009. doi:10.1007/978-3-642-11269-0_6.

11 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discret. Appl. Math.*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

12 Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.

13 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

14 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.

15 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *CoRR*, abs/1103.0534, 2011. arXiv:1103.0534.

16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

17 Guillaume Ducoffe. Maximum matching in almost linear time on graphs of bounded clique-width. *Algorithmica*, 84(11):3489–3520, 2022. doi:10.1007/s00453-022-00999-9.

18 Fedor V. Fomin and Tuukka Korhonen. Fast fpt-approximation of branchwidth. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 886–899. ACM, 2022. doi:10.1145/3519935.3519996.

19 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

20 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Representative families of product families. *ACM Trans. Algorithms*, 13(3):36:1–36:29, 2017. doi:10.1145/3039243.

21 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

22 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

23 Falko Hegerfeld and Stefan Kratsch. Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.29.

24 Falko Hegerfeld and Stefan Kratsch. Towards exact structural thresholds for parameterized complexity. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPIcs*, pages 17:1–17:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.17.

25 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by clique-width. *CoRR*, abs/2302.03627, 2023. doi:10.48550/arXiv.2302.03627.

26 Falko Hegerfeld and Stefan Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. *CoRR*, abs/2302.14128, 2023. doi:10.48550/arXiv.2302.14128.

27 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

28 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

29 Yoichi Iwata and Yuichi Yoshida. On the equivalence among problems of bounded width. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 754–765. Springer, 2015. doi:10.1007/978-3-662-48350-3_63.

30 Hugo Jacob, Thomas Bellitto, Oscar Defrain, and Marcin Pilipczuk. Close relatives (of feedback vertex set), revisited. In Petr A. Golovach and Meirav Zehavi, editors, *16th International Symposium on Parameterized and Exact Computation, IPEC 2021, September 8-10, 2021, Lisbon, Portugal*, volume 214 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.21.

31 Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. *Theor. Comput. Sci.*, 795:520–539, 2019. doi:10.1016/j.tcs.2019.08.006.

32 Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discrete Applied Mathematics*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

33 Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. doi:10.1007/BFb0045375.

34 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

**35** Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

**36** Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

**37** Jesper Nederlof. Algorithms for np-hard problems via rank-related parameters of matrices. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms - Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2020. doi:10.1007/978-3-030-42071-0_11.

**38** Jesper Nederlof, Michal Pilipczuk, Céline M. F. Swennenhuis, and Karol Wegrzycki. Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science - 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020. doi:10.1007/978-3-030-60440-0_3.

**39** Jesper Nederlof, Johan M. M. van Rooij, and Thomas C. van Dijk. Inclusion/exclusion meets measure and conquer. *Algorithmica*, 69(3):685–740, 2014. doi:10.1007/s00453-013-9759-2.

**40** Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

**41** Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM Trans. Comput. Theory*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

**42** Willem J. A. Pino, Hans L. Bodlaender, and Johan M. M. van Rooij. Cut and count and representative sets on branch decompositions. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 27:1–27:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.27.

**43** Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

# Pareto Sums of Pareto Sets

**Demian Hespe**
Karlsruhe Institute of Technology, Germany

**Peter Sanders**
Karlsruhe Institute of Technology, Germany

**Sabine Storandt**
University of Konstanz, Germany

**Carina Truschel**
University of Konstanz, Germany

## Abstract

In bi-criteria optimization problems, the goal is typically to compute the set of Pareto-optimal solutions. Many algorithms for these types of problems rely on efficient merging or combining of partial solutions and filtering of dominated solutions in the resulting sets. In this paper, we consider the task of computing the Pareto sum of two given Pareto sets $A, B$ of size $n$. The Pareto sum contains all non-dominated points of the Minkowski sum $M = \{a + b | a \in A, b \in B\}$. Since the Minkowski sum has a size of $n^2$, but the Pareto sum $C$ can be much smaller, the goal is to compute $C$ without having to compute and store all of $M$. We present several new algorithms for efficient Pareto sum computation, including an output-sensitive one with a running time of $\mathcal{O}(n \log n + nk)$ and a space consumption of $\mathcal{O}(n + k)$ for $k = |C|$. We also describe suitable engineering techniques to improve the practical running times of our algorithms and provide a comparative experimental study. As one showcase application, we consider preprocessing-based methods for bi-criteria route planning in road networks. Pareto sum computation is a frequent task in the preprocessing phase. We show that using our algorithms with an output-sensitive space consumption allows to tackle larger instances and reduces the preprocessing time compared to algorithms that fully store $M$.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Theory of computation → Randomness, geometry and discrete structures

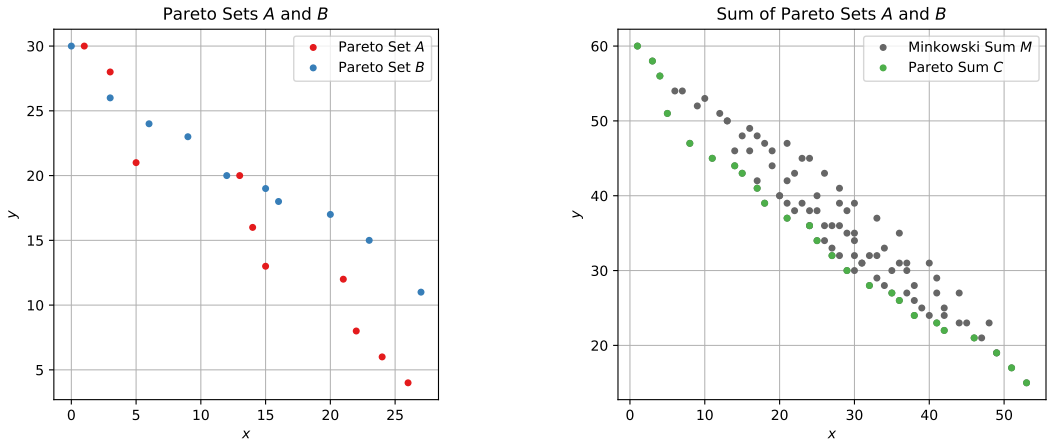
**Keywords and phrases** Minkowski sum, Skyline, Successive Algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.60

**Funding** *Peter Sanders*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).
*Carina Truschel*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 251654672 – TRR 161.

## 1 Introduction

Solving multi-objective combinatorial optimization problems demands to find the set of non-dominated solutions, also referred to as skyline, Pareto frontier or Pareto set. To solve problem instances of substantial size, solution approaches often rely on efficient combination and filtering of partial solutions. In particular, non-dominance filtering of unions or Minkowski sums of intermediate Pareto sets occur as a frequent subtasks in optimization algorithms. Examples include decomposition approaches for multi-objective integer programming [16], dynamic programming methods for multi-objective knapsack [7], bi-directional search algorithms for multi-criteria shortest path problems [4], or Pareto local search for multi-objective set cover [14].




| | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $A_9$ | $A_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $B_1$ | 1, 60 | 3, 58 | 5, 51 | 13, 50 | 14, 46 | 15, 43 | 21, 42 | 22, 38 | 24, 36 | 26, 34 |
| $B_2$ | 4, 56 | 6, 54 | 8, 47 | 16, 46 | 17, 42 | 18, 39 | 24, 38 | 25, 34 | 27, 32 | 29, 30 |
| $B_3$ | 7, 54 | 9, 52 | 11, 45 | 19, 44 | 20, 40 | 21, 37 | 27, 36 | 28, 32 | 30, 30 | 32, 28 |
| $B_4$ | 10, 53 | 12, 51 | 14, 44 | 22, 43 | 23, 39 | 24, 36 | 30, 35 | 31, 31 | 33, 29 | 35, 27 |
| $B_5$ | 13, 50 | 15, 48 | 17, 41 | 25, 40 | 26, 36 | 27, 33 | 33, 32 | 34, 28 | 36, 26 | 38, 24 |
| $B_6$ | 16, 49 | 18, 47 | 20, 40 | 28, 39 | 29, 35 | 30, 32 | 36, 31 | 37, 27 | 39, 25 | 41, 23 |
| $B_7$ | 17, 48 | 19, 46 | 21, 39 | 29, 38 | 30, 34 | 31, 31 | 37, 30 | 38, 26 | 40, 24 | 42, 22 |
| $B_8$ | 21, 47 | 23, 45 | 25, 38 | 33, 37 | 34, 33 | 35, 30 | 41, 29 | 42, 35 | 44, 23 | 46, 21 |
| $B_9$ | 24, 45 | 26, 43 | 28, 36 | 36, 35 | 37, 31 | 38, 28 | 44, 27 | 45, 23 | 47, 21 | 49, 19 |
| $B_{10}$ | 28, 41 | 30, 39 | 32, 32 | 40, 31 | 41, 27 | 42, 24 | 48, 23 | 49, 19 | 51, 17 | 53, 15 |

**Figure 1** Example instance with input Pareto sets $A, B$ of size 10. The Minkowski sum has 100 elements. The Pareto sum $C$ consists of 27 elements (marked green in the plot as well as in the matrix representation).

In this paper, we focus on the efficient computation of the filtered Minkowski sum of two-dimensional Pareto sets $A, B$. The Minkowski sum $M$ is defined as the set of elements derived from pairwise addition of elements in $A$ and $B$. However, the Minkowski sum often contains many dominated elements. In fact, it was proven in [12] that for $A, B$ of size $n$, the set of non-dominated elements in $M$ – which we refer to as Pareto sum of $A, B$ – might have a size in $o(n)$. Thus, algorithms that first compute all elements of $M$ and subsequently apply non-dominance filtering might be unnecessarily wasteful as they come with a running time and space consumption in $\Omega(n^2)$. The goal of the paper is to design practical algorithms for Pareto sum computation with output-sensitive space consumption, and to evaluate their performance on realistic inputs. As one particular use case of our methods, we will consider the bi-criteria route planning problem in road networks. There exists a plethora of algorithms to compute the set of Pareto-optimal paths between a given source and a target node in the network, see e.g. [8, 2]. The currently fastest methods rely on preprocessing. In particular, variants of contraction hierarchies (CH) have been proven to be very useful in this context [17, 19]. In a CH, the input graph is augmented with so called shortcut edges that represent sets of Pareto-optimal paths between their end points. The shortcuts store the costs of these paths in the form of Pareto sets. On query time, shortcuts are instrumented to decrease the search space size of a Pareto-Dijkstra run, resulting in significantly faster query times and reduced space consumption. In the preprocessing phase, the shortcuts are inserted incrementally. The base operation is to concatenate two shortcut or original edges $e = \{u, v\}$ and $e' = \{v, w\}$ to form a new shortcut $\{u, w\}$. The Pareto set of the new shortcut is the set of non-dominated elements in the Minkowski sum of the Pareto sets corresponding to $e$ and $e'$. Thus, the preprocessing time crucially depends on an efficient Pareto sum computation. In [9],

it was discussed that computing the Minkowski sum and filtering all dominated elements in a naive fashion is too time-consuming. Therefore, filtering strategies were proposed that prune dominated elements. However, these strategies are not guaranteed to retrieve the Pareto sum but usually produce a superset thereof. Keeping supersets slows down later stages of the preprocessing as well as query answering. We will propose novel algorithms that allow for fast and exact Pareto sum computation.

## 1.1 Related Work

Non-dominance filtering in point sets is a well-studied task in computational geometry [6], also referred to as skyline or maxima computation. There exist output-sensitive algorithms for the two-dimensional case as e.g. the one proposed by Kirkpatrick and Seidel [11] with a running time of $\mathcal{O}(N \log k)$ where $N$ denotes the size of the point set and $k$ the size of the skyline. The basic idea is to first partition the input points into $k$ sets of size $\approx N/k$ with non-overlapping ranges with respect to their $x$-coordinates. Then, the sets are processed individually in sorted order. As $k$ is typically not known beforehand, a more intricate version of the algorithm allows to achieve the same asymptotic running time by starting with a coarse partition and refining it on demand as soon as a certain number of non-dominated points are identified. With a worst-case running time of $\mathcal{O}(N \log N)$ and close-to-linear running time for small $k$, this algorithm seems to be well-suited for Pareto sum computation. However, in our application we have $N = |M|$ where $M$ is the Minkowski sum of the input sets $A, B$; and any approach that relies on access to $M$ as a whole is bound to a running time and space consumption in $\Omega(n^2)$.

Using the interpretation of input elements as two-dimensional points, Pareto sum computation can also be reduced to computing the Minkowski sum of the orthogonal hulls of $A$ and $B$ (where both sets are augmented with a dummy point based on the maximum coordinate values in the respective set). The Minkowski sum of two convex polygons can be computed in linear time [15]. For non-convex inputs $P, Q$, the polygons are first decomposed into convex subpolygons $P_1, \ldots, P_s$ and $Q_1, \ldots, Q_t$. Then, the linear time algorithm is applied to all pairs $P_i, Q_j$, and finally the union of all partial results is computed. The running time depends on the applied decomposition technique and the number and complexity of the resulting subpolygons [1]. However, if $P$ and $Q$ are orthogonal convex hulls of size $n$, their convex decomposition cannot contain fewer than $n$ subpolygons, and thus the approach needs to compute the union of $\Theta(n^2)$ partial solutions.

Recently, new algorithms for Pareto sum computation with the potential to achieve subquadratic running time and space consumption were proposed in [12]. The so called NonDomDC algorithm exploits the structure of the matrix that represents the Minkowski sum (see Figure 1). In particular, it makes use of the fact that columns in the matrix are Pareto sets themselves. Assuming the Pareto sum $P_i$ of elements occurring in the first $i$ columns is known, $P_{i+1}$ can be computed by merging $P_i$ and column $i+1$ and pruning dominated elements in $\mathcal{O}(|P_i| + n)$ time. Thus, with $P := \max_{i=1}^{n} |P_i|$ denoting the maximum size of an intermediate solution, the total running time is in $\mathcal{O}(Pn)$ and the space consumption is in $\mathcal{O}(n + P)$. However, this does not constitute an output-sensitive algorithm as the size of the intermediate Pareto sum can be significantly larger than the final result size. So even for small $k$, the algorithm might have cubic running time and quadratic space consumption. However, their experimental study demonstrates good performance in practice. Similar methods, as the box-based method proposed in [10], were shown to be outperformed.

**Table 1** Running time and space consumption of different algorithms for Pareto sum computation. The input size is denoted by $n$ and the output size by $k$.

| algorithm | | running time | space | |
|---|---|---|---|---|
| NonDomDC | (ND) | $\mathcal{O}(n^3)$ | $\mathcal{O}(n^2)$ | [12] |
| Kirkpatrick-Seidel | (KS) | $\mathcal{O}(n^2 \log k)$ | $\Theta(n^2)$ | [11] |
| Brute Force | (BF) | $\mathcal{O}(n^4)$ | $\mathcal{O}(n+k)$ | 4.1 |
| Binary Search | (BS) | $\mathcal{O}(n^3 \log n)$ | $\mathcal{O}(n+k)$ | 4.2 |
| Sort & Compare | (SC) | $\mathcal{O}(n^2 \log n)$ | $\mathcal{O}(n+k)$ | 4.3 |
| Successive Binary Search | (SBS) | $\mathcal{O}(nk \log n)$ | $\mathcal{O}(n+k)$ | 5.1 |
| Successive Sweep Search | (SSS) | $\mathcal{O}(n \log n + nk)$ | $\mathcal{O}(n+k)$ | 5.2 |

## 1.2 Contribution

In this paper, we consider the problem of efficient Pareto sum computation in theory and practice. First, we present an algorithm that has the ability to identify a subset of the Pareto sum $C$ in linear time. This algorithm can be used as a preprocessing step for all other approaches for Pareto sum computation. Additionally, we show that for certain kinds of inputs, the algorithm already returns whole set $C$. Then, we present and thoroughly analyze several algorithms for Pareto sum computation with a special focus on achieving an output-sensitive space consumption. Table 1 provides an overview of the characteristics of our proposed algorithms as well as existing baseline approaches. In an extensive experimental study, we compare their scalability. We consider randomly generated data as well as real inputs that stem from bi-criteria route planning instances. For both input types alike, our output-sensitive successive sweep search proves to be the most efficient algorithm. This aligns well with our theoretical analysis, as it turns out, especially for large input sizes, that the Pareto sum $C$ contains only a small fraction of the elements in the Minkowski sum.

# 2 Problem Definition

In this section, we formally define the notion of a Pareto sum and provide notation used throughout the paper.

▶ **Definition 1** (Domination). *Given two points $p, p' \in \mathbb{R}^2$, we say that $p$ dominates $p'$, or $p \prec p'$, if $p \neq p'$ and $p.x \leq p'.x$ as well as $p.y \leq p'.y$.*

▶ **Definition 2** (Pareto set). *A set $S \subset \mathbb{R}^2$ is a Pareto set if no point in $S$ dominates another point in $S$, that is $\nexists s, s' \in S$ with $s \prec s'$.*

We always assume that Pareto sets are sorted in lexicographic order. We use $S_i$ to refer to the element with rank $i$ in set $S$.

▶ **Definition 3** (Minkowski sum). *Given two Pareto sets $A, B \subset \mathbb{R}^2$, their Minkowski sum $M = A \oplus B$ is defined as $M := \{a + b |\ a \in A, b \in B\}$.*

In a slight abuse of notation, we will use $M$ to refer to the set of elements in the Minkowski sum as well as the matrix where $M_{ij} = A_i + B_j$.

▶ **Definition 4** (Pareto sum). *Let $A, B \subset \mathbb{R}^2$ be two Pareto sets of size $n$ and let $M = A \oplus B$ denote their Minkowski sum. Then the Pareto sum $C$ of $A, B$ is defined as the set of all non-dominated points in $M$.*

Figure 1 illustrates the concepts of Minkowski and Pareto sums. Throughout the paper, we will use $k$ to denote the size of $C$.

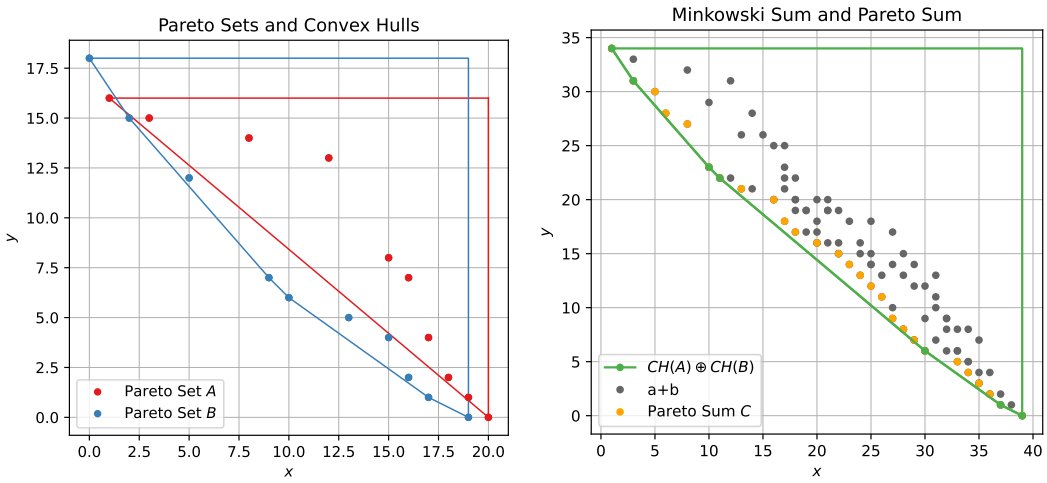



**Figure 2** Left: Pareto sets $A, B$ together with their convex hulls. Right: The Minkowski sum $CH(A) \oplus CH(B)$ of the convex hulls encloses all pairwise vector additions $a + b$ with $a \in CH(A)$ and $b \in CH(B)$. The respective vertices (green points) are a subset of the Pareto Sum $C$.

## 3 Minkowski Sum of Convex Hulls

In this section, we present an algorithm that for given Pareto sets $A, B$ computes part of their Pareto sum in linear time. Thus, this algorithm can be used as an efficient preprocessing method before applying other (more costly) techniques.

For two convex polygons $P, Q \in \mathbb{R}^2$, their Minkowski sum $P \oplus Q$ is a convex polygon with at most $|P| + |Q|$ vertices and these vertices can be computed in linear time [15]. Let now $A, B$ be sorted Pareto sets augmented with dummy points $(x, y)$ where $x := \max_{s \in S} s.x$ and $y := \max_{s \in S} s.y$ for $S = A$ and $S = B$, respectively. We use $CH(A)$ and $CH(B)$ to refer to the convex hulls of these two sets. The following observation captures the connection between these convex hulls and the Pareto sum.

▶ **Observation 5.** *The vertices of the Minkowski sum $CH(A) \oplus CH(B)$ are a subset of the Pareto sum of $A$ and $B$ (excluding the dummy point sum).*

For a sorted Pareto set, its convex hull can be computed in linear time using Andrew's algorithm [3]. Then, using the linear time Minkowski sum algorithm on the two convex hulls and extracting the respective polygon vertices, we obtain a subset of the Pareto sum $C$, see Figure 2. If both $A, B$ are convex, then this procedure already returns all of $C$. We thus get the following corollary.

▶ **Corollary 6.** *The Pareto sum of two convex, sorted Pareto sets can be computed in $\mathcal{O}(n)$.*

For non-convex $A, B$, we might only get part of the Pareto sum. However, as this step only takes linear time (assuming the Pareto sets are presorted), it can always be used as an initial step before applying other algorithms. We will discuss below in more detail how the knowledge of $C' \subset C$ can be exploited to decrease the practical running time of several of the algorithms we propose.

# 4 Base Algorithms

In this section, we discuss three simple base algorithms for Pareto sum computation along with engineering concepts for their acceleration and space consumption reduction. The algorithms all proceed by checking for each element $p \in M$ whether there exists $p' \in M$ that dominates $p$. If there is no such $p'$, the point $p$ is added to the Pareto sum $C$. The only difference between the algorithms is the implementation of the dominance check.

## 4.1 Brute Force (BF)

The easiest way to check for a point $p \in M$ whether it is non-dominated is by pairwise comparison to all other elements in $M$. This dominance check takes $\mathcal{O}(|M|)$ time per point, accumulating to a total time of $\mathcal{O}(|M|^2) = \mathcal{O}(n^4)$. As the elements $M_{ij}$ can be computed on demand, the space consumption is linear in the input size $n$ and the output size $k$.

▶ **Corollary 7.** *The BF algorithm runs in $\mathcal{O}(n^4)$ time using $\mathcal{O}(n+k)$ space.*

## 4.2 Binary Search (BS)

To decrease the time needed for the dominance check, we take the structure of $M$ into account. Based on the assumption that $A$ and $B$ are sorted and that $M_{ij}$ is defined as $A_i + B_j$, we have the property that each column (and each row) of the matrix $M$ forms a sorted Pareto set on its own. Thus, if we want to check whether column $M_j$ contains an element dominating $p$, we simply have to find the entry $M_{ij}$ with the largest index $i$ such that $M_{ij}.x \leq p.x$ as well as the entry $M_{i'j}$ with the smallest index $i'$ such that $M_{i'j}.y \leq p.y$. Then all elements in $M_j$ with a row index in $[i', i]$ dominate $p$ (or are equal to $p$). Accordingly, the dominance check for $M_j$ boils down to evaluating whether $i' \leq i$ holds. These two indices can each be identified via a binary search over the respective coordinates in the column. Hence the dominance check time per column is in $\mathcal{O}(\log n)$, resulting in a total time of $\mathcal{O}(n \log n)$ per element in $M$. Entries of $M$ that need to be accessed can be computed on demand.

▶ **Corollary 8.** *BS runs in $\mathcal{O}(n^3 \log n)$ time using $\mathcal{O}(n+k)$ space.*

To reduce the practical running time of the BS algorithm, we propose the following engineering techniques.

**Pruning.** Whenever we identify a non-dominated point $p$ and add it to $C$, we can also compute all entries in $M$ dominated by $p$ in time $\mathcal{O}(n \log n)$, again with the help of two binary searches per column. For those points, dominance does not need to be checked again. However, if we simply store a flag for each entry in $M$ whether it needs to be further considered or not, the space consumption increases to $n^2$. Instead we can store for each column the set of intervals of dominated points in an interval tree. The number of intervals per column is upper bounded by $k$. Then, for a point $p = M_{ij}$ we can query the interval tree in time $\mathcal{O}(\log k)$ to see whether the point lies in a dominated region. Intervals can also be added or merged within the same time. However, the space consumption would still increase to $\mathcal{O}(nk)$. To keep the space consumption linear, one might only want to store a constant number of intervals per column (e.g. only the largest one) and then merge or replace intervals if possible or needed. If pruning is applied, we can also disregard fully dominated columns in the binary searches of the remaining elements.

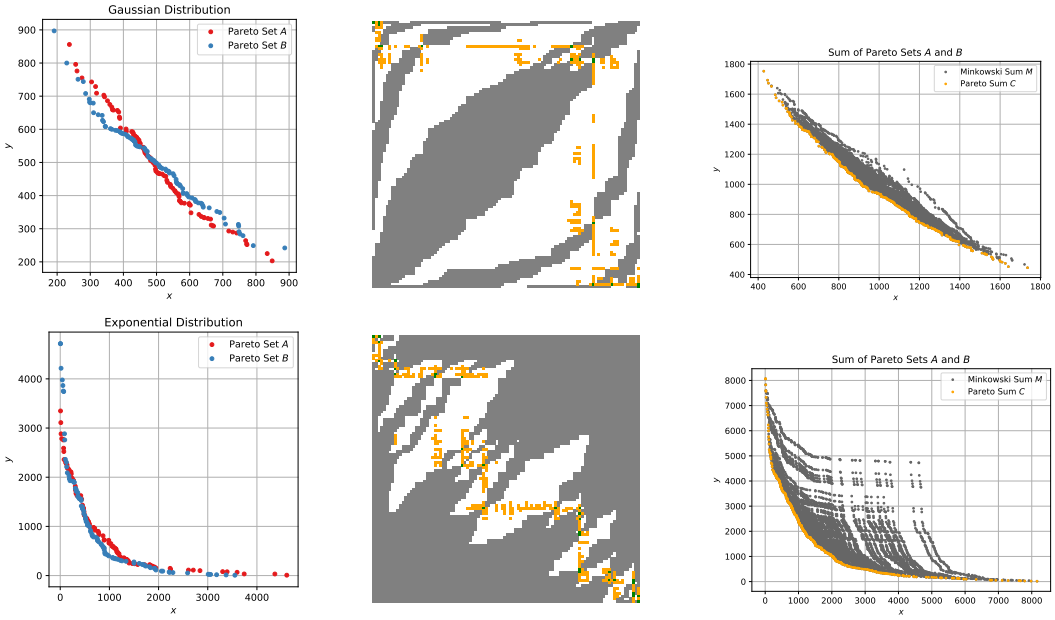



**Figure 3** Input Pareto sets following different kinds of distributions (left) and schematic depiction of the corresponding Minkowski matrix $M$ (middle). Green dots indicate entries in $M$ that are points on the Minkowski sum of the convex hulls of $A$ and $B$, black dots indicate entries that are dominated by the green ones, and orange dots encode the remaining elements of the Pareto sum which then together dominate the white dots. In the right images, the Minkowski sum and the Pareto sum are illustrated based on point coordinates.

**Priority Binary Search (PBS).** As soon as dominance checks might be avoided for some of the elements based on the pruning techniques described above, the order in which the points are considered impacts the running time. Identifying points that dominate many other points early on can significantly reduce the total number of checks. For that purpose, we will use the preprocessing step described in Section 3 to get an initial set of points $C' \subset C$. We can directly exclude any points dominated by the points in $C'$. Furthermore, we conjecture that points in the same rows or columns as the points of $C'$ occupy in $M$ are likely to be also part of $C$. Thus, we give priority to these points in our search. Figure 3 shows some visual support for this hypothesis.

## 4.3 Sort & Compare (SC)

Given a sorted set of points, extracting the set of non-dominated points can be accomplished in constant time per point. The smallest element is always added to $C$. For each other element in sorted order, we check whether it is dominated by (or equal to) the currently last element in $C$. If that is not the case, the element is added to $C$.

Computing and sorting $M$ as a whole takes time $\mathcal{O}(n^2 \log n)$ and requires quadratic space. But we can exploit the structure of $M$ to improve the space consumption to linear as follows: We use a min-heap data structure and initialize it with the first row of $M$. Each element in the heap remembers its position in $M$. When we extract the min element $M_{ij}$ from the heap and $C$ is empty so far, we add the element to $C$. Otherwise, we compare $M_{ij}$ to the element added to $C$ last. If $M_{ij}$ is not dominated, we also add it to $C$. In any case, we add its column successor $M_{i+1j}$ to the heap (as long as $i < n$). As each column is a sorted Pareto

set in itself, we know that $M_{i+1j}$ has to have larger $x$-value than $M_{ij}$. Thus, we extract the elements from the heap exactly according to their global lexicographic order. As the heap never contains more than $n$ elements, its space consumption is in $\mathcal{O}(n)$ and the heap operations take $\mathcal{O}(\log n)$ per round.

In conclusion, the heap-based variant has the same asymptotic running time as the one where we fully compute and sort $M$, but a significantly reduced space consumption.

▶ **Corollary 9.** *SC runs in $\mathcal{O}(n^2 \log n)$ time using $\mathcal{O}(n+k)$ space.*

## 5 Output-Sensitive Algorithms

If the Pareto sum $C$ contains (almost) all elements of the Minkowski sum $M$, a quadratic running time is needed already to report $C$. In this case, the running time of SC is asymptotically optimal up to logarithmic factors. However, in case $C$ is small, subquadratic running times might be possible. We will present two output-sensitive algorithms in this section that have a running time asymptotically faster than SC for $k \in o(n)$ or $k \in o(n \log n)$, respectively. Both algorithms detect the elements in $C$ successively. This is a well-established paradigm for output-sensitive skyline computation, see e.g. [13, 18]. However, known algorithms rely on the explicit availability of the point set to construct an efficient search data structure. Based on the following lemma, we will design successive algorithms that do not need access to $M$ as a whole.

▶ **Lemma 10.** *Let $A, B$ be two Pareto sets and $c, c' \in C$ two elements of their Pareto sum with $c.x < c'.x$. Then the lexicographically smallest element $m$ in $M$ that dominates $(c'.x - \varepsilon, c.y - \varepsilon)$ for $\varepsilon > 0$ is also part of $C$ (if such an element exists).*

**Proof.** We first argue that for any $m \in M$, the smallest point $p \in M$ dominating $m$ (or being equal to $m$) is part of the Pareto sum $C$. Assume otherwise for contradiction. Then there is a point $p' \in C$ that dominates $p$ and thus also $m$. But in this case $p'$ is smaller than $p$ which contradicts the choice of $p$ as smallest element to dominate $m$.

Now, if there is any point $m \in M$ that dominates the dummy point $(c'.x - \varepsilon, c.y - \varepsilon)$ the above argumentation applies. ◀

Clearly, $M_{11}$ and $M_{nn}$ are always part of the Pareto sum, as those are the points with smallest global $x$-value and $y$-value, respectively. All other elements in $C$ must have an $x$-value in the open interval $(M_{11}.x, M_{nn}.x)$. Thus, if we have an oracle that returns the smallest point $m \in M$ (with respect to lexicographic ordering) in a given range $[x_{\min}, x_{\max}) \times [y_{\min}, y_{\max})$, we can compute $C$ based on Lemma 10 as follows. We initialize $C = M_{11}, M_{nn}$ and $x_{\min} = M_{11}.x, x_{\max} = M_{nn}.x, y_{\min} = M_{nn}.y, y_{\max} = M_{11}.y$. Then we query the oracle to get the smallest point $m$ in the respective range. If such a point does not exist, we abort. Otherwise we add the point $m$ to $C$ and set $x_{\min} = m.x, y_{\max} = m.y$ before repeating the process. Figure 4 illustrates the core concept.

Thus, the algorithm discovers the points in $C$ one-by-one in increasing order of their $x$-values (except for $M_{nn}$ which is known from the start) using $k$ calls to the range-minimum oracle. A naive oracle implementation would be to check all points in $M$ for containment in the range and to keep track of the minimum among them. Then each call to the oracle costs $\mathcal{O}(n^2)$ and the overall running time of the successive algorithm would be $\mathcal{O}(n^2 k)$. Next, we describe how to implement the oracle more efficiently.

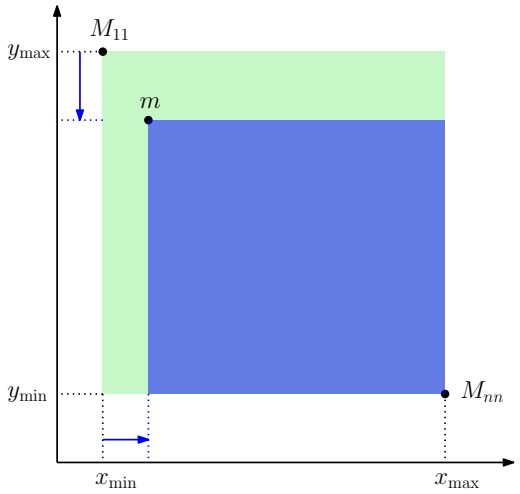



**Figure 4** Initial search range (green rectangle) spanned by $M_{11}$ and $M_{nn}$. The range-minimum element $m$ then leads to a reduction of $y_{\max}$ and an increase of $x_{\min}$ (blue arrows) which tightens the search range for the next element of $C$.

## 5.1 Successive Binary Search (SBS)

In the BS approach described in Section 4.2, we use two binary searches per column of $M$ to check for a query point $m \in M$ whether an element dominating $m$ exists in that column. We can use the same concept to implement a range-minimum oracle: For each column, we identify via binary searches the first position $f_x$ with an $x$-coordinate larger or equal to $x_{\min}$, the last position $l_x$ with an $x$-coordinate smaller than $x_{\max}$, the first position $f_y$ with a $y$-coordinate smaller than $y_{\max}$, and the last position $l_y$ with a $y$-coordinate larger or equal to $y_{\min}$. If $[f_x, l_x] \cap [f_y, l_y] \neq \emptyset$, we return $\max(f_x, f_y)$. The entry at that position is the smallest point dominating $m$ in the column. Keeping track of the smallest returned point over all columns provides the desired result in $\mathcal{O}(n \log n)$ per oracle call.

▶ **Corollary 11.** *Successive BS runs in $\mathcal{O}(nk \log n)$ time using $\mathcal{O}(n + k)$ space.*

For standard BS we can use the Minkowski sum of the convex hulls of $A$ and $B$ to already identify a subset $C'$ with size $k' \leq k$ of the Pareto sum $C$ without the need of binary searches. We can apply the same initialization here and then simply use the successive algorithm independently in each of the $k' - 1$ ranges induced by any two consecutive points in $C'$ (in sorted order). By that, the number of oracle calls increases to at most $k + k' - 1 < 2k$ as in each of the $k - 1$ ranges the respective last oracle call will return no point. This does not affect the asymptotic running time, though. But it allows to conduct up to $k' - 1$ oracle calls in parallel.

To further foster parallelization, we can weaken the oracle requirement to always return the smallest point in a given range to the requirement to return any non-dominated point in the range. The new point then splits the previous range into two subranges that can be queried independently. With that, the oracle might be called up to $k + 2k' - 2 < 3k$ times as now in each of the $k' - 1$ ranges the call to its leftmost induced subrange and the call to its rightmost induced subrange will return no point. Again, the asymptotic running time remains unaffected. To implement the weaker oracle, we propose the so called Cascading BS (CBS) algorithm. Again, we consider the columns one after each other. But now, as soon as we find a column entry $p$ in the given query range, we update the range immediately and then search for a point dominating $p$ in the remaining columns. If we find such a point,

we immediately update again. Note that it can never happen that a point $p'$ in an already visited column dominates $p$, as then we would have selected one point from said column to tighten the query range and there can never be two points in one column dominating one another. Thus, after we considered all columns, we can safely add the current point $p$ to $C$.

For an example of the difference between SBS and CBS, consider the matrix in Figure 1 and assume the current search range is $[14, 53) \times [15, 44)$. SBS discovers the element $A_6 + B_1 = (15, 43)$ next as this is the point with smallest $x$-value that dominates the dummy point $(53 - \varepsilon, 44 - \varepsilon)$. CBS, however, first detects the point $A_1 + B_{10} = (28, 41)$ as it already dominates the dummy point. It then proceeds by trying to find an element that dominates $(28, 41)$. It thus detects $A_3 + B_5 = (17, 41)$ next and tightens the search range accordingly. As this is a Pareto sum point, no further dominating elements are found in the remaining columns and $(17, 41)$ is returned and added to $C$. The search range is then split into $[14, 17) \times [41, 44)$ and $[17, 53) \times [15, 41)$, which can be processed independently.

## 5.2 Successive Sweep Search (SSS)

To improve the oracle time of SBS, we observe that the binary searches in the columns are somewhat redundant. If $M$ was fully available, we could apply fractional cascading [5] to the column vectors. This would reduce the running time to compute the positions $f_x, l_x, f_y, l_y$ in all columns from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$. Thus, the $k$ oracle calls cost $\mathcal{O}(nk)$. Unfortunately, computing $M$ and the data structure for fractional cascading requires space and time in $\Theta(n^2)$. Fortunately, we can also achieve linear oracle time without the need to access $M$ as a whole. Based on the structure of $M$, we know that for an entry $M_{ij}$ all elements $M_{st}$ with $s \geq i$ and $t \geq j$ have a larger $x$-coordinate than $M_{ij}$ but a smaller $y$-coordinate. Vice versa, all elements in $M_{st}$ with $s \leq i$ and $t \leq j$ have a smaller $x$-coordinate than $M_{ij}$ but a larger $y$-coordinate. This implies, for example, that the position of $f_x$ in some column cannot be larger than the position of $f_x$ in the neighboring column to its left. Similar relationships hold for the positions of $l_x, f_y$ and $l_y$ in neighboring columns. Accordingly, we can find the respective column values by a single left-to-right sweep, where the search path forms a monotone staircase structure and is thus bounded in length by $2n$.

Even better, we can have a single unified sweep to find the range-minimum $m$ in linear time: We start at $M_{n1}$, that is, the last entry of the first column. Whenever we enter a new column $j$, we apply upwards linear search in that column until we reach an entry $M_{ij}$ where either $M_{ij}.x > x_{\min}$ and $M_{i-1j}.x \leq x_{\min}$ or where $M_{ij}.y < y_{\max}$ and $M_{i-1j}.y \geq y_{\max}$. Thus, we get $i = \max(f_x, f_y)$; except if the entry we start from already has a too small $x$-value or a too large $y$-value or both which means that the column contains no point in the query range. In the former case, we check whether $M_{ij}$ is contained in the range. If the check is passed, $M_{ij}$ is a valid candidate for the range-minimum $m$. We keep track of the smallest viable candidate over the course of the algorithm. We then go from element $M_{ij}$ to its right neighbor $M_{ij+1}$ and proceed with the new column as described above. After processing the last column, we return the current $m$ as the range-minimum element.

▶ **Lemma 12.** *The sweep algorithm computes the smallest $m \in M$ in a given range in $\mathcal{O}(n)$.*

**Proof.** To prove correctness, we need to argue that for a column $j$ entered at row $i$ and exited at row $i' \leq i$, the range-minimum $m$ can not be an entry $M_{i^*j}$ with $i^* < i'$ or $i^* > i$.

Clearly, checking elements in column $j$ with a row index smaller than $i'$ cannot give us any viable candidates, as either their respective $x$-value is too small or their $y$-value is too large by definition. Hence we only need to consider $i^* > i$. If $M_{ij}$ is in the query range, then the entries in column $j$ with $i^* > i$ cannot constitute the range-minimum as they all have an $x$-value larger than that of $M_{ij}$. If $M_{ij}$ is not in the query range, we have the following cases:

- $M_{ij}.x > x_{\max}$. But then $M_{i^*j}.x > x_{\max}$ holds as well.
- $M_{ij}.x < x_{\min}$ or $M_{ij}.y > y_{\max}$. This case only occurs if $i = n$. Thus there is no $i^* > i$.
- $M_{ij}.y < y_{\min}$. But then $M_{i^*j}.y < y_{\min}$ holds as well.

Accordingly, if column $j$ contains the range minimum it needs to be an element with a row index in $[i', i]$. If $M_{i'j}$ is in the range, it is clearly the best candidate in column $j$ for the range-minimum $m$. If $M_{i'j}$ is not in the query range, then the same applies to all entries in the same column with larger row index as argued above. Thus, it is sufficient to check $M_{i'j}$ for each column $j$. The running time is determined by the number of elements in $M$ that are accessed. As the interval of elements checked for each column only overlaps with the intervals of all columns to its left in a single row index, at most $2n$ elements in $M$ are considered in total. ◀

Based on this sweep search (SS) oracle, we now get a successive algorithm with better running time than SBS.

▶ **Corollary 13.** *Successive SS runs in $\mathcal{O}(n \log n + nk)$ using $\mathcal{O}(n + k)$ space.*

In fact, if $k \in o(\log n)$, the running time is dominated by the initial sorting step of the elements in $A, B$. For $k \in o(n)$, we achieve a subquadratic running time.

For acceleration of sweep search in practice, we observe that if we enter a column at row $i$ and confirm for some value $i' < i$ that $M_{i'j}$ is still feasible with respect to $x_{\min}$ and $y_{\max}$, we do not have to check intermediate rows to get the correct range-minimum by virtue of Lemma 12. Similarly, if we have not found any $M_{ij}$ in column $j$ with $M_{ij}.x \geq x_{\min}$ and $M_{ij}.y < y_{\max}$ and the same inequalities apply to some $M_{ij'}$ with $j' > j$, we do not need to check intermediate columns for range-minimum candidates. Thus, in both cases we can introduce a skip threshold $\Delta > 1$ and check for $i' = i - \Delta$ or $j' = j + \Delta$, respectively, whether the necessary conditions apply. If that is the case we skip intermediate rows or columns and then try to skip ahead again. If skipping is no longer possible, we simply fall back to linear search. Accordingly, in the worst case, we check at most one superfluous element for each row and column. This does not increase the asymptotic running time of the oracle but might reduce its running time in practice if skipping is successful.

Furthermore, similar to CBS, we also propose Cascading Sweep Search (CSS). Here again, whenever we found a temporary range-minimum candidate $m$ we immediately tighten the search range to enforce that further candidates need to dominate the current $m$ to be considered. The sweep search then also guarantees to return an element of the Pareto sum $C$. The Minkowski hull preprocessing and the split of search intervals to foster parallelization as described for CBS can be applied here as well.

## 6 Experimental Evaluation

We implemented the seven algorithms for Pareto sum computation listed in Table 1 in C++: The two existing approaches, namely the Kirkpatrick-Seidel algorithm (KS) [11] and NonDomDC (ND) [12], the three base algorithms (BF, BS, SC), and the two successive algorithms (SBS, SSS). For KS, we implemented the simpler (and thus faster) variant, where the output size $k$ (used for partitioning) is given as an input. We simply compute $k$ with one of our other algorithms and then feed the result into KS. For ND, we actually implemented two variants described in [12]: In the first variant (described in more detail in Section 1.1), one always merges the current result with the next column (Sequential ND, SND). In the second variant, columns are combined in a MergeSort like fashion (Doubling ND, DND). As benchmark data we use randomly generated inputs as well as real inputs. Both types of data sets are described in more detail below. All experiments were conducted on a single core of a 3.4 GHz AMD Ryzen Threadripper 1950X 16-core processor with 126 GB of RAM.

## 6.1 Results for Generated Data

To generate Pareto sets, we take two random samples of $n$ unique values from a given range. The first sample is sorted increasingly and represents the $x$-coordinates within the Pareto set. The second sequence is sorted decreasingly and represents the $y$-coordinates. We consider uniform, Gaussian and exponentially distributed samples over the range $[0, n]$. In addition, we investigate $A$ and $B$ where the respective $x$-coordinates are drawn from vastly different intervals, namely with upper bounds $\sqrt{n}$ and $n^2$. We call this a shifted distribution. Figures 3 and 5 show example instances for each input type. Running times are always averaged over 100 generated instances per tested value of $n$.

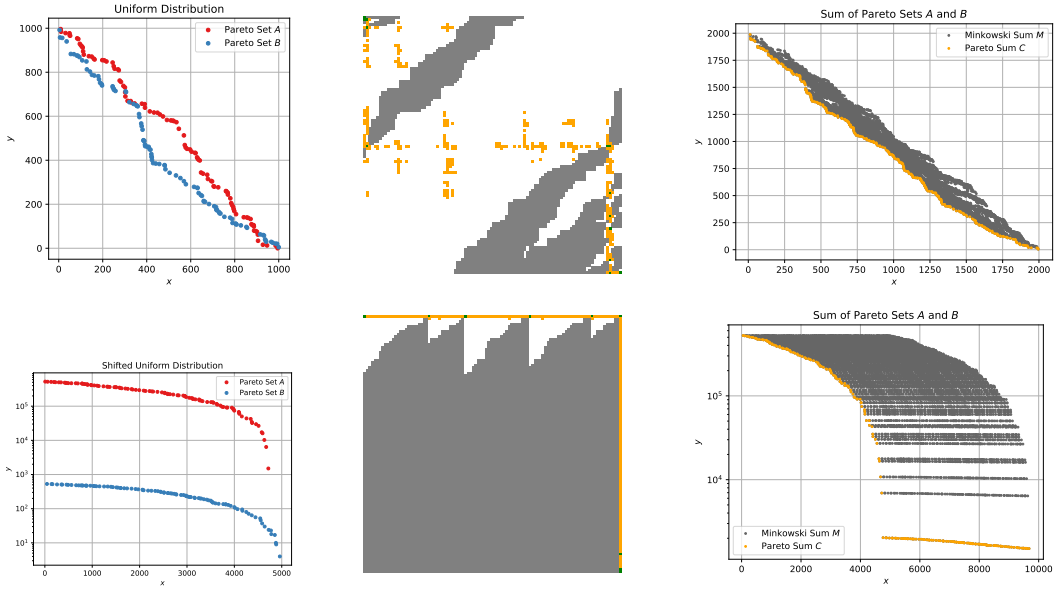



**Figure 5** Example instances for uniform and shifted uniform point distributions For the latter, note the logscale of the y-axis. The color coding is the same as in Figure 3.

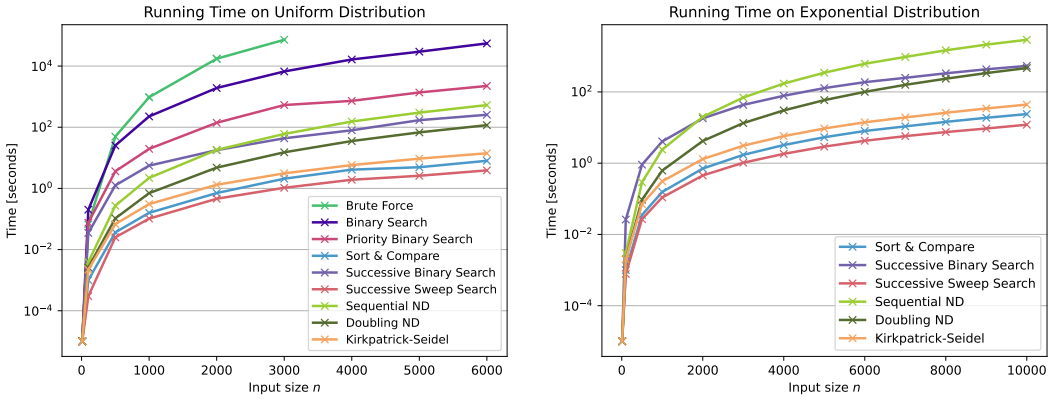



**Figure 6** Average running times of all algorithms on uniform distributions (left) and of the top six algorithms on exponential distributions (right). Note the logscale of the y-axis.

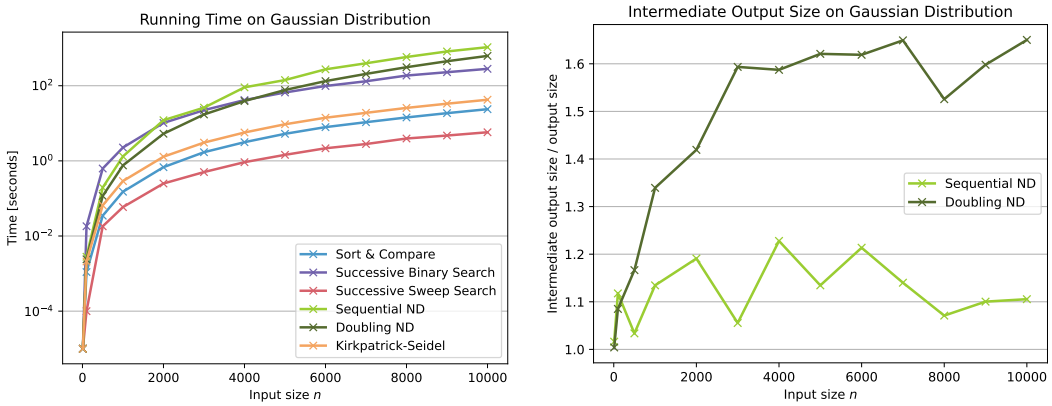



**Figure 7** Left: Average running times of selected algorithms on Gaussian distribution. Right: Intermediate output size of the Sequential ND and Doubling ND algorithms.

Figure 6, left, shows the running times of all algorithms on uniformly distributed instances. In line with our theoretical analysis, the Brute Fore approach is by far the slowest. Instances larger than $n = 3000$ were not tested as those already took over an hour. The engineered Binary Search algorithm (PBS) is faster than BS by an order of magnitude but not as fast as the other competitors. Note that we used the variant here that guarantees output-sensitive space consumption by storing the borders of at most on block of dominated elements per column. In compliance with the experimental results in [12], we see that DND is faster than SND. However, they are both slower than the Kirkpatrick-Seidel and the Sort & Compare algorithm, which exhibit very similar running times. SSS is faster than the second best algorithm, Sort & Compare, by a factor of 2-5. On exponential distributions the results are similar, see Figure 6, right. But the ND variants perform slightly worse. On Gaussian distributions, the ND variants are about two orders of magnitude slower than SSS and even scale worse than SBS, see Figure 7, left. The reason for this behavior is investigated in Figure 7, right, which depicts the intermediate solution sizes of the ND algorithms. On uniform and exponential distributions, the space overhead is less than 1%. However, on Gaussian distributions, SND and DND require 20% and 65% more space than the output (and our output-sensitive algorithms), respectively. Also note that even if intermediate sizes are not much larger than the final size, it might be that the maximum intermediate size is reached early and persists close to that value, thereby increasing the running times of the individual column merge steps. Thus, the ND algorithms are both very sensitive to the distribution of the input points and the position of the Pareto sum points within the matrix. In contrast, the performance of our sweep algorithms depends primarily on the size of the output, which was within $4n$ across all tested instances and distributions.

On uniform, Gaussian and exponential distributions, our engineered SSS variant with $\Delta$-skipping had little impact. On shifted distributions, however, this concept proved to be very effective due to the Pareto sum points being mostly located either in the first few rows or the last few columns of the matrix (see Figure 5), and Pareto sum sizes being small in general. Using $\Delta = \sqrt{n}$, we achieved speeds-ups of two orders of magnitude over all other approaches on instances with $n = 10000$.

**Table 2** Experimental results for BCH computation on three road networks of different size. The table shows the input graph sizes, the number of edges in the augmented graph (original + shortcuts), the number of non-trivial Pareto sum computations, as well as the running times for conducting these computations with four different algorithms. The final row shows the preprocessing time spent on operations other than Pareto sum computation.

| | ROAD1 | ROAD2 | ROAD3 |
|---|---|---|---|
| #Nodes | 349479 | 1246440 | 3835238 |
| #Edges | 720363 | 2612260 | 8037228 |
| #BCH-edges | 1325259 | 4981957 | 15703653 |
| #PS computations | 899390 | 4424857 | 48844050 |
| Kirkpatrick-Seidel | 32.09 s | 1234.15 s | >24 h |
| Sort & Compare | 13.45 s | 587.77 s | 47374.67 s |
| Doubling ND | 19.03 s | 454.42 s | 37447.61 s |
| Successive Sweep Search | 4.83 s | 129.62 s | 9668.33 s |
| Additional preprocessing | 7.23 s | 54.66 s | 1237.58 s |

## 6.2 Results for Real Data

As a real-world application of Pareto sum computation, we consider bi-criteria route planning in road networks. Here, given an input graph $G(V, E)$ and costs $c_1, c_2 : E \to \mathbb{R}^+$, the goal is to either compute all Pareto-optimal paths with respect to $c_1, c_2$ between two nodes $s, t \in V$, or the path optimal with respect to one cost while not exceeding a budget on the other (also known as the constrained shortest path problem). To accelerate query answering, a bi-criteria contraction hierarchy (BCH) data structure can be used. In the preprocessing phase of a BCH, the input graph is augmented with additional edges, also called shortcuts. The shortcut insertion is guided by a node permutation $\pi : V \to \{1, \dots, n\}$. For nodes $u, w \in V$, a shortcut $\{u, w\}$ is inserted if and only if there exists a simple path from $u$ to $w$ on which no node has a higher $\pi$ value than $\max(\pi(u), \pi(w))$. The shortcut represents all simple paths $p$ between $u$ and $w$ with that property. For each Pareto-optimal $p$, the respective cost tuple $(c_1(p), c_2(p))$ should be assigned to the shortcut. To compute these Pareto sets for all shortcuts in an efficient manner, a bottom-up approach is used. Let $u$ be the inner node on a path $p$ from $u$ to $w$ with maximum $\pi$-value. If the Pareto sets $A$ and $B$ of the shortcuts $\{u, v\}$ and $\{v, w\}$ are known, respectively, the Pareto set of $\{u, w\}$ is the Pareto sum $C$ of $A$ and $B$. If there are multiple paths $p$, the final Pareto set of $\{u, w\}$ is formed by the non-dominated elements of the union of all these Pareto sums. The non-dominated union of two Pareto sets can be computed in linear time by merging the presorted sets to obtain the sorted union and then applying the simple non-dominance check as described in the SC approach. In the final BCH, queries can be answered with a bi-directional Pareto-Dijkstra run that relaxes shortcut edges instead of many original edges whenever possible. This significantly reduces the search space and allows to answer queries orders of magnitude faster [17, 9].

In our experiments, we use test graphs extracted from OpenStreetMap with Euclidean distance and positive height difference as edge costs (in compliance with [17]). Based on our results on generated data, we use the best four algorithms (KS, SC, DND and SSS) for Pareto sum computation. Note that Pareto sets $A$ and $B$ do not necessarily have the same size here, but all proposed Pareto sum computation algorithms can be easily adapted. Table 2 shows the characteristics of the three road network instances we considered in our experiments and the outcomes. The number of Pareto sum computations in the preprocessing phase of the BCH reported in the table excludes trivial inputs where either $A$ or $B$ has size 1. We

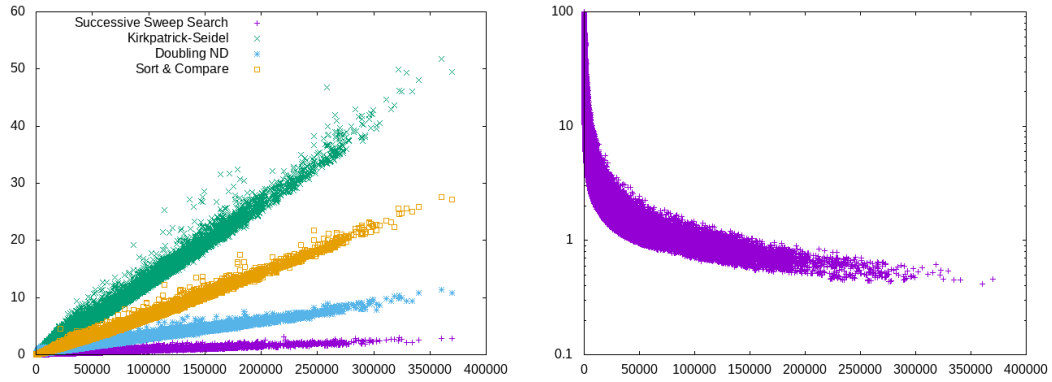



**Figure 8** Detailed results for the ROAD2 instance. Left: Running times in seconds per Pareto sum computation in dependency of the size of the Minkowski sum. Right: Pareto sum size as percentage (logscale) of the size of the Minkowski sum.

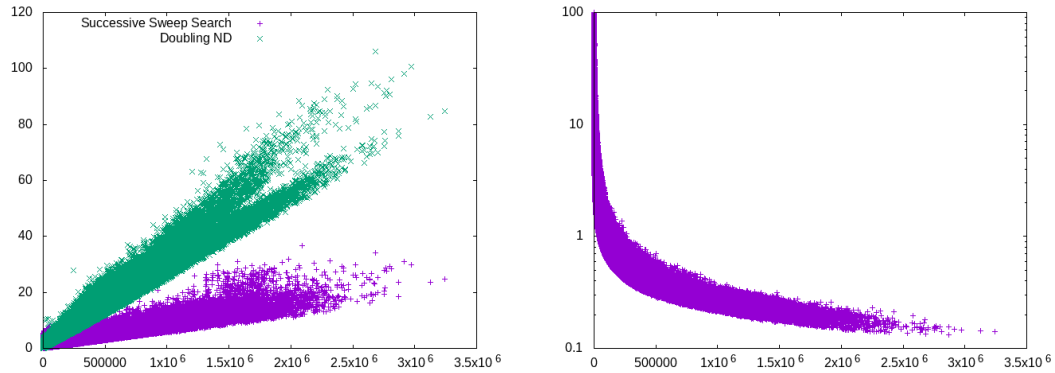



**Figure 9** Detailed results for the ROAD3 instance. Left: Running times in seconds per Pareto sum computation in dependency of the size of the Minkowski sum. Right: Pareto sum size as percentage (logscale) of the size of the Minkowski sum.

observe that the time spent on non-trivial Pareto sum computations dominates the overall preprocessing time, especially on larger networks. There are significant differences in running time between the algorithms we tested, though. On all instances, SSS is the fastest approach. It is roughly an order of magnitude faster than the KS algorithm which fully computes and stores $M$. With KS, we could not compute a BCH data structure within a day on our largest instance with about 4 million nodes. Interestingly, in contrast to the experiments on generated data, DND outperforms SC. Figure 8 shows the running times for all individual Pareto sum computations as well as the size of the respective results for the ROAD2 instance. We observe that the larger the Minkowski sum $M$, the smaller the relative output size. This explains why SSS consistently outperforms the other approaches, especially on larger inputs. Figure 9 shows results for the two best algorithms, DND and SSS, on ROAD3. Here, $|M|$ was up to $3 \cdot 10^6$ and the percentage of elements in the Pareto sum $C$ even approached 0.1. This is very beneficial for the SSS algorithm as the smaller the output size the fewer range minimum oracle calls are needed.

Furthermore, we used DND and SSS in query answering to combine Pareto sets in the bi-directional Pareto-Dijkstra run whenever the forward and the backward search meet. On the ROAD3 instance, a speed-up of up to 5 over DND was achieved when using SSS.

## 7 Conclusions and Future Work

We introduced scalable algorithms for Pareto sum computation which avoid the computation of the whole Minkowski sum. Our successive sweep search algorithm was shown to perform best across all instances, generated or real, while guaranteeing an output-sensitive space consumption. One direction for future work is to carefully parallelize all discussed algorithms. We also implemented and tested the cascading sweep search variant, which enables parallel successive search, and observed that the sequential running time matches that of successive sweep search while splitting the search ranges in many subranges which could be processed in parallel. Furthermore, even in a parallel implementation, the sweep search algorithm keeps its output-sensitive space consumption. Another direction for future work is the consideration of higher-dimensional input points. While some algorithms are easily generalizable, novel range minimum oracles need to be designed for the successive algorithms to work.

## References


1 Pankaj K Agarwal, Eyal Flato, and Dan Halperin. Polygon decomposition for efficient construction of minkowski sums. *Computational Geometry*, 21(1-2):39–61, 2002.

2 Saman Ahmadi, Guido Tack, Daniel D Harabor, and Philip Kilby. Bi-objective search with bi-directional a*. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 142–144, 2021.

3 Alex M Andrew. Another efficient algorithm for convex hulls in two dimensions. In *Information Processing Letters*, volume 9.5, pages 216–219. Elsevier, 1979.

4 Christian Artigues, Marie-José Huguet, Fallou Gueye, Frédéric Schettini, and Laurent Dezou. State-based accelerations and bidirectional search for bi-objective multi-modal shortest paths. *Transportation Research Part C: Emerging Technologies*, 27:233–259, 2013.

5 Bernard Chazelle and Leonidas J Guibas. Fractional cascading: A data structuring technique with geometric applications. In *Automata, Languages and Programming: 12th Colloquium Nafplion, Greece, July 15–19, 1985*, pages 90–100. Springer, 2005.

6 Wei-Mei Chen, Hsien-Kuei Hwang, and Tsung-Hsi Tsai. Maxima-finding algorithms for multidimensional samples: A two-phase approach. *Computational Geometry*, 45(1-2):33–53, 2012.

7 Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-spektrum*, 22:425–460, 2000.

8 Stephan Erb, Moritz Kobitzsch, and Peter Sanders. Parallel bi-objective shortest paths using weight-balanced b-trees with bulk updates. In *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 111–122. Springer, 2014.

9 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2015.

10 Antoine Kerbérénès, Daniel Vanderpooten, and Jean-Michel Vanpeperstraete. Computing efficiently the nondominated subset of a set sum. *International Transactions in Operational Research*, 2022.

11 David G Kirkpatrick and Raimund Seidel. Output-size sensitive algorithms for finding maximal vectors. In *Proceedings of the First Annual Symposium on Computational Geometry*, pages 89–96, 1985.

12 Kathrin Klamroth, Bruno Lang, and Michael Stiglmayr. Efficient dominance filtering for unions and minkowski sums of non-dominated sets. *Available at SSRN 4308273*, 2022.

13 Jinfei Liu, Li Xiong, and Xiaofeng Xu. Faster output-sensitive skyline computation algorithm. *Information Processing Letters*, 114(12):710–713, 2014.

**14** Thibaut Lust and Daniel Tuyttens. Variable and large neighborhood search to solve the multiobjective set covering problem. *Journal of Heuristics*, 20:165–188, 2014.

**15** de Berg Mark, Cheong Otfried, van Kreveld Marc, and Overmars Mark. *Computational geometry algorithms and applications*. Spinger, 2008.

**16** Britta Schulze, Kathrin Klamroth, and Michael Stiglmayr. Multi-objective unconstrained combinatorial optimization: a polynomial bound on the number of extreme supported solutions. *Journal of Global Optimization*, 74(3):495–522, 2019.

**17** Sabine Storandt. Route planning for bicycles—exact constrained shortest paths made practical via contraction hierarchy. In *Twenty-second international conference on automated planning and scheduling*, 2012.

**18** Chih-Chiang Yu, Wing-Kai Hon, and Biing-Feng Wang. Improved data structures for the orthogonal range successor problem. *Computational Geometry*, 44(3):148–159, 2011.

**19** Han Zhang, Oren Salzman, Ariel Felner, TK Satish Kumar, Carlos Hernández Ulloa, and Sven Koenig. Efficient multi-query bi-objective search via contraction hierarchies. In *Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS)*, 2023.

# Solving Edge Clique Cover Exactly via Synergistic Data Reduction

**Anthony Hevia**
Hamilton College, Clinton, NY, USA

**Benjamin Kallus**
Dartmouth College, Hanover, NH, USA

**Summer McClintic**
Hamilton College, Clinton, NY, USA

**Samantha Reisner**
Hamilton College, Clinton, NY, USA

**Darren Strash**[1]
Hamilton College, Clinton, NY, USA

**Johnathan Wilson**
Hamilton College, Clinton, NY, USA

## Abstract

The edge clique cover (ECC) problem – where the goal is to find a minimum cardinality set of cliques that cover all the edges of a graph – is a classic NP-hard problem that has received much attention from both the theoretical and experimental algorithms communities. While small sparse graphs can be solved exactly via the branch-and-reduce algorithm of Gramm et al. [JEA 2009], larger instances can currently only be solved inexactly using heuristics with unknown overall solution quality. We revisit computing minimum ECCs exactly in practice by combining data reduction for both the ECC *and* vertex clique cover (VCC) problems. We do so by modifying the polynomial-time reduction of Kou et al. [Commun. ACM 1978] to transform a reduced ECC instance to a VCC instance; alternatively, we show it is possible to "lift" some VCC reductions to the ECC problem. Our experiments show that combining data reduction for both problems (which we call *synergistic data reduction*) enables finding exact minimum ECCs orders of magnitude faster than the technique of Gramm et al., and allows solving large sparse graphs on up to millions of vertices and edges that have never before been solved. With these new exact solutions, we evaluate the quality of recent heuristic algorithms on large instances for the first time. The most recent of these, EO-ECC by Abdullah et al. [ICCS 2022], solves 8 of the 27 instances for which we have exact solutions. It is our hope that our strategy rallies researchers to seek improved algorithms for the ECC problem.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Packing and covering problems

**Keywords and phrases** Edge clique cover, Vertex clique cover, Data reduction, Degeneracy

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.61

**Related Version** *Full Version*: `https://arxiv.org/abs/2306.17804`

**Supplementary Material** *Software (Source Code)*: `https://github.com/darrenstrash/Redu3ECC`

**Acknowledgements** We thank the anonymous reviewers for their insightful feedback, David Swartz from Hamilton College for technical support, and Adam Chrisman, Caitlin Matwijec-Walda, and Jon Matwijec-Walda for a cozy space to work at The Copper Easel and Superofficial in Rome, NY.

[1] Corresponding author.

# 1 Introduction

In the *edge clique cover (ECC) problem*, also called the *clique cover* problem, we are given an unweighted, undirected, simple graph $G = (V, E)$ and asked to find a minimum cardinality set of cliques that cover the edges of $G$. The ECC problem is NP-hard, however its decision variant did not appear in Karp's original list of NP-complete problems [23], though the *vertex clique cover (VCC) problem* did. Compared to the VCC problem, the ECC problem has received the lion's share of attention from researchers, in part because it has many applications. For instance, edge clique covers can be used to succinctly represent constraints for integer program solvers [5] and to detect communities in networks [12].

Data reduction rules, which allow one to transform an input instance to a smaller equivalent instance of the same problem, are powerful tools for solving NP-hard problems in practice [4, 26]. Of particular interest in the field of parameterized algorithms is whether the repeated application of data reduction rules produces a *kernel* – which is a problem instance that has size bounded by a function $O(f(k))$ of some parameter $k$ of the input. Gramm et al. [19] show that repeated application of four simple reduction rules produce a kernel of size $2^k$, where the parameter $k$ is the number of cliques in the cover. When intermixed with branch-and-bound (a so-called *branch-and-reduce* algorithm), these reduction rules enable solving sparse graphs of up to 10,000 vertices quickly in practice. Since their seminal work, no progress has been made on solving larger instances exactly. Indeed, the prospect of doing so is grim since polynomial kernels are unlikely to exist for the ECC problem, when parameterized on the solution size [13]. Although researchers have found further FPT algorithms (and smaller kernels) with other parameters [6, 32], these algorithms are still only able to solve relatively small instances in practice. The outlook for the VCC problem is even worse in theory: it is unlikely to have any problem kernel when parameterized on the number of cliques $k$ in the cover, as it is already NP-hard for $k = 3$ (since it is equivalent to 3-coloring the complement graph).

However, recent data reductions for the VCC problem have been shown to significantly accelerate computing minimum VCCs exactly in practice. Strash and Thompson [31] introduce a suite of reduction rules and show that data reduction can solve real-world sparse graphs with up to millions of vertices in seconds.

## Our Results

We show that combining VCC and ECC data reductions enables the ECC problem to be solved exactly on large instances not previously solvable by Gramm et al. [19]. We do so by modifying the polynomial-time transformation of Kou et al. [25] to transform a reduced ECC instance to a VCC instance, but also show that some VCC data reductions can be "lifted" to ECC data reductions. Their combined reduction power (which we call *synergistic data reduction*) reduces an ECC instance significantly more than Gramm et al.'s reductions alone, enabling us to exactly solve graphs with millions of vertices and edges. With these exact results, we objectively evaluate the quality of heuristic algorithms recently introduced in the literature. On instances not solvable exactly with our method, we give upper and lower bounds for use by future researchers.

# 2 Related Work

We now briefly review the relevant previous work on the ECC and VCC problems, as well as practical data reduction in related problems.

## 2.1 Edge Clique Cover

The goal of the edge clique cover (ECC) problem is to cover the edges of the graph $G$ with a minimum number of cliques, denoted $\theta_E(G)$. That is, to find a set of cliques $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ such that each edge is in at least one clique in $\mathcal{C}$ and $k = \theta_E(G)$. Although closely related to the VCC problem (to cover *vertices* with a minimum number of cliques, denoted $\theta(G)$), Brigham and Dutton [8] showed that $\theta(G) \leq \theta_E(G)$, and that these cover numbers can differ significantly: $\theta_E(G)$ can be as large as $\theta(G)(n - \theta(G))$. Gramm et al. [19] introduced four data reductions for the ECC problem, which they show can solve real-world sparse graphs of hundreds of vertices, as well as synthetic instances on up to 10K vertices in practice, when interleaved with branch and bound. Furthermore, they showed that their data reductions produce a kernel of size $2^k$, where $k$ is the number of cliques. Cygan et al. [13] showed that it is unlikely that a polynomial-size kernel exists when parameterized by the number of cliques in the cover, as otherwise the polynomial hierarchy collapses to its third level. However, Blanchette et al. [6] gave a linear-time algorithm having running time $O(2^{\binom{k}{2}}n)$ where $k$ is the treewidth of the graph. In practice, their algorithm is effective on graphs with hundreds of vertices and small treewidth. For larger graphs, heuristic methods are used to compute inexact ECCs [12, 2, 1] in practice. No heuristic algorithm performs best on all instances, and their overall quality is unclear.

## 2.2 Vertex Clique Cover

The vertex clique cover (VCC) problem is NP-hard, and closely related to the maximum independent set and graph coloring problems. The size of a minimum VCC (also called the clique cover number) $\theta(G)$ is lower bounded by the size of a maximum independent set (the independence number $\alpha(G)$) and equivalent to the chromatic number of the complement graph, $\chi(\overline{G})$. There is a rich line of research on the graph coloring problem, which seeks to compute the chromatic number; many of the theoretical results for the VCC problem come via the graph coloring problem. The fastest exact exponential-space algorithm for computing the chromatic number on an $n$-vertex graph has time $O^*(2^n)$ (where $O^*$ hides polynomial factors) using a generalization of the exclusion-inclusion principle [24], and in polynomial space the problem can be solved in time $O(2.2356^n)$ [18]. Furthermore, there exists no polynomial-time algorithm with approximation ratio better than $n^{1-\epsilon}$ for $\epsilon > 0$ unless $P = NP$ [34].

In terms of data reduction, we note that it is unlikely that a kernel exists when parameterized on the (vertex) clique cover number. Deciding if a cover with even 3 cliques exists is NP-complete (since 3-coloring the complement is NP-hard). A polynomial kernel would have size $O(1)$ and could be computed in polynomial time. Solving the kernel with brute-force computation would solve the VCC problem in polynomial time, implying $P = NP$. However, in practice, the VCC problem can be solved on large, sparse real-world graphs using the data reductions by Strash and Thompson [31].

## 2.3 Data Reduction in Practice for Related Problems

Other classical NP-hard problems have large suites of data reductions that are effective in practice, including minimum vertex cover [4, 15], maximum cut [16], and cluster editing [7]. Popular data reductions include variations of simplicial vertex removal, degree-2 folding, twin, domination, unconfined, packing, crown, and linear-programming-relaxation-based reductions [4]. Even the simplest reductions can be highly effective when combined with

other techniques [10, 30]. Data reductions are most effective in sparse graphs, which are the graphs that we consider here. Finally, similar to what we propose here, other NP-hard problems are solved by first applying a problem transformation. In particular, algorithms for minimum dominating set problem first transform the problem to an instance of the set cover problem [33].

## 3 Preliminaries

We consider a simple finite undirected graph $G = (V, E)$ with vertex set $V$ and edge set $E \subseteq \{\{u, v\} \mid u, v \in V\}$. For brevity, we denote by $n = |V|$ and $m = |E|$ the number of vertices and edges in the graph, respectively. When more specificity is needed, we denote the vertex and edge set of a graph $G$ by $V(G)$ and $E(G)$ respectively. We say two vertices $u, v \in V$ are *adjacent* (or *neighbors*) when $\{u, v\} \in E$. The *open neighborhood* of a vertex $v \in V$ is the set of its neighbors $N(v) := \{u \mid \{u, v\} \in E\}$, and the *degree* of $v$ is $|N(v)|$. We further define the *closed neighborhood* of a vertex $v \in V$ to be $N[v] := N(v) \cup \{v\}$. Extending these definitions, the open neighborhood of a set $A \subseteq V$ is $N(A) := \bigcup_{v \in A} N(v) \setminus A$ and the closed neighborhood of $A$ is $N[A] := \bigcup_{v \in A} N[v]$. The subgraph of $G$ induced by a vertex set $V' \subseteq V$, denoted $G[V']$, has vertex set $V'$ and edge set $E' = \{\{u, v\} \in E \mid u, v \in V'\}$. The *degeneracy* $d$ of a graph $G$ is the smallest value such that every nonempty subgraph of $G$ has a vertex of degree at most $d$ [27]. It is possible to order the vertices of a graph $G$ in time $O(n + m)$ so that every vertex has $d$ or fewer neighbors later in the ordering; such an ordering is called a *degeneracy* ordering [14].

A vertex set $C \subseteq V$ is called a *clique* if, for each pair of distinct of vertices $u, v \in C$, $\{u, v\} \in E$. A set of cliques $\mathcal{C}$ is called an *edge clique cover (ECC)* (or just a *clique cover*) of $G$ if for every edge $\{u, v\} \in E$ there exists at least one $C \in \mathcal{C}$ such that $\{u, v\} \subseteq C$. That is, there is some clique in $\mathcal{C}$ that *covers* $\{u, v\}$. The set of cliques $\mathcal{C}$ is said to cover the graph $G$. An ECC of minimum cardinality is called a *minimum ECC*, and its cardinality is denoted by $\theta_E(G)$, called the edge clique cover number.

Similarly, in a *vertex clique cover (VCC)*, every *vertex* $v \in V$ is covered by some clique. The cardinality of a minimum VCC is the *clique cover number*, denoted by $\theta(G)$.

## 4 Existing Tools Discussion

In this section, we discuss basic tools that we will use to solve the ECC problem, together with insights into their behavior on sparse graphs. We begin by describing the existing ECC data reductions by Gramm et al. [19]. We then discuss how to convert an input ECC instance to an equivalent VCC instance using the technique of Kou et al. [25]. We will extend these tools to develop our full algorithm combining ECC and VCC reductions in the next section.

### 4.1 ECC Reduction Rules

Gramm et al. [19] introduce four data reduction rules that either cover edges by a clique known to be in a minimum cardinality ECC or add edges to the input graph $G$. Once all of a vertex $v$'s incident edges are covered, $v$ can be removed from the graph.

With each edge $\{u, v\}$, Gramm et al. store the common neighbors in $G$, denoted by $N_{\{u,v\}}$, as well as a count $c_{\{u,v\}} = |E(G[N_{\{u,v\}}])|$ of the edges between common neighbors. These values are updated in ECC Reduction 1 and are used in ECC Reduction 2.

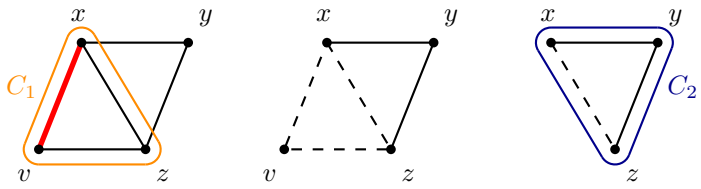



**Figure 1** Illustrating Gramm et al. [19]'s data reductions: (left) edge $\{v, x\}$ is in exactly one maximal clique $C_1$, triggering ECC Reduction 2 and covering edges $\{v, x\}$, $\{v, z\}$, and $\{x, z\}$ (middle). Vertex $v$ can then be removed with ECC Reduction 1. The remaining triangle (right) is covered by clique $C_2$ by applying ECC Reduction 2 to either $\{x, y\}$ or $\{y, z\}$.

Throughout the application of data reductions, vertices are removed from $G$ and edges are covered. Figure 1 illustrates an example of the reductions. Set let edge set $E' \subseteq E$ be the set of uncovered edges (by extension, $E \setminus E'$ are the covered edges). The graph $G$ only changes when a vertex is removed.

We note that the data reductions by Gramm et al. [19] are particularly effective for sparse graphs; however, the original data reductions were not written with efficiency in mind. Although these reductions have (very) slow theoretical running times, we offer insights as to why their reductions are faster in practice than indicated by the theoretical running time from Gramm et al. [19].

▶ **ECC Reduction 1** ([19]). *Let $v \in V$ be a vertex whose incident edges are all covered (i.e., in $E \setminus E'$). Then remove $v$ from the graph $G$, along with its incident edges, and update values $c_{\{w,x\}}$ and $N_{\{w,x\}}$ for all uncovered edges $\{w, x\} \in E'$ whose endpoints are both adjacent to $v$, i.e., $\{w, x\} \subseteq N(v)$.*

As noted by Gramm et al. [19], this step can be applied to all vertices in running time $O(n^2m)$ by iterating over each vertex $v$ and updating $N_{\{u,w\}}$ for all edges $\{u, w\} \in E'$ whose endpoints are adjacent to $v$. However, in sparse graphs the maximum degree in $G$, denoted $\Delta$, is significantly smaller than $n$. Each edge $\{u, w\}$ has its set $N_{\{u,w\}}$ updated at most $\Delta$ times, taking $O(\Delta)$ time to update each time, giving a more reasonable running time of $O(\Delta^2 m)$. We note that with adjustments, this can be run faster by enumerating all triangles in $G$ in time $O(dm)$ using the triangle listing by Chiba and Nishizeki [11] and updating $N_{\{u,w\}}$ for edge $\{u, w\}$ in each triangle; however, this is a different implementation than that done by Gramm et al. [19] and not our focus here.

▶ **ECC Reduction 2** ([19]). *Let edge $\{u, v\} \in E'$ be an uncovered edge such that $c_{\{u,v\}} = \binom{|N_{\{u,v\}}|}{2}$ (i.e., the edge is in exactly one maximal clique in $G'$). Then $C = N_{\{u,v\}} \cup \{u, v\}$ is a maximal clique of $G$ in some minimum ECC. Add the clique $C$ to the clique cover, and cover any uncovered edges in $C$ in $G$.*

As noted by Gramm et al. [19], ECC Reduction 2 can be implemented in time $O(n^2m)$ by iterating over each edge $\{u, v\} \in E'$, checking if $c_{\{u,v\}} = \binom{|N_{\{u,v\}}|}{2}$ in $O(1)$ time, and covering the edges of $\{u, v\}$'s clique in time $O(n^2)$ time.

However, when run on sparse graphs, which tend to have low degeneracy $d$ [14], this rule is much faster. Graphs with degeneracy $d$ have cliques of at most $d + 1$ vertices, therefore the reduction is only triggered when $|N_{\{u,v\}}| < d$. Hence, in practice, we should observe the much faster running time of $O(d^2m)$.

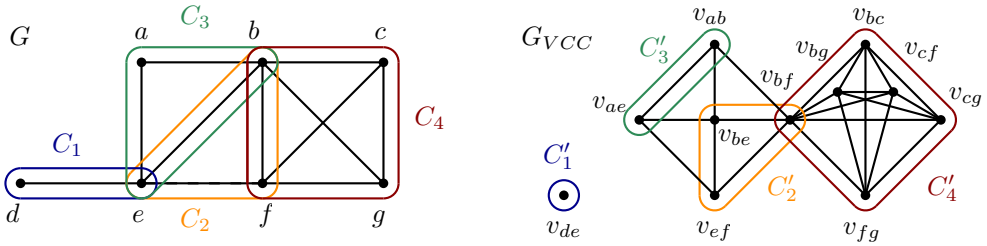



**Figure 2** An example graph $G$ with minimum ECC and its transformed graph $G_{VCC}$ with a corresponding minimum VCC.

Gramm et al. introduce two more ECC reductions (which we'll refer to as ECC Reductions 3 and 4), however, they are more complex and less effective than ECC Reductions 1 and 2 in practice. Experiments by Gramm et al. show that these reductions are very slow, and only improve the search tree size by a constant factor when incorporated into branch and reduce [19]. We therefore omit them from our discussion and implementation.

## 4.2 Transforming an ECC Instance to a VCC Instance

Kou et al. [25] showed that the ECC problem is NP-hard via a polynomial-time reduction from the VCC problem. Furthermore, they gave a polynomial-time reduction *to* the VCC problem, which we use as the basis of our transformation. We describe their transformation and briefly justify why it works.

Given an input graph $G = (V, E)$ for the ECC problem Kou et al. [25] transform $G$ to a new graph $G_{VCC} = (V_{VCC}, E_{VCC})$ that is an equivalent VCC instance as follows. For each edge $\{x, y\} \in E$, create a new vertex $v_{xy} \in V_{VCC}$, then add an edge $\{v_{xy}, v_{wz}\}$ to $E_{VCC}$ if and only if there exists a clique $C$ in $G$ containing both $\{x, y\}$ and $\{w, z\}$. Now, for any given subset $C \subset V_{VCC}$, $C$ is a clique in $G_{VCC}$ iff its vertices' corresponding edges in $E$ also induce a clique in $G$. Hence, a minimum cardinality VCC in $G_{VCC}$ corresponds to a minimum cardinality ECC in $G$. (See Figure 2.)

To determine if two edges are in a clique together in $G$, Kou et al. [25] make the following observation:

▶ **Observation 1** ([25]). *Two distinct edges $\{x, y\}, \{w, z\}$ are in a clique together in $G$ iff $\{x, y\}$ and $\{w, z\}$ are incident and $\{x, y\} \cup \{w, z\}$ induce a triangle, or $\{x, y\}$ and $\{w, z\}$ are not incident and $\{w, x, y, z\}$ form a 4-clique.*

However, there is a clear issue when using this transformation: $G_{VCC}$ can be very large. We briefly discuss its size and sparsity.

### 4.2.1 The Effect of Transformation on Graph Size and Sparsity

In the worst case, the size of $G_{VCC}$ is a quadratic factor larger than $G$. Indeed, if the graph $G$ is itself the complete graph $K_n$, on $n$ vertices and $\Theta(n^2)$ edges, then the transformed graph is the complete graph $K_{n(n-1)/2}$ having $\Theta(n^2)$ nodes and $\Theta(n^4)$ edges. However, we show that the size of the graph only increases by a factor of $O(d^2)$, where $d$ is the degeneracy of the graph. Real-world sparse graphs have low degeneracy [14], and thus this is a significant improvement over the worst case.

▶ **Theorem 2.** *Let the degeneracy of $G = (V, E)$ be $d$. Then $|V_{VCC}| = m \leq dn$ and $|E_{VCC}| = O(d^2 m)$.*

**Proof.** By construction $|V_{VCC}| = m$; hence, to bound $|V_{VCC}|$, we bound the number of edges in $G$. In a degeneracy ordering of the graph, each vertex has at most $d$ later neighbors in the ordering. Therefore, $|V_{VCC}| = m \leq dn$. To bound $|E_{VCC}|$, we compute an upper bound on the number of triangles and 4-cliques in $G$. Following Observation 1, each edge in $E_{VCC}$ corresponds to a pair of edges in $E$ contained in a triangle or a pair of non-incident edges in a 4-clique. Each triangle has 3 edges, and each 4-clique has 3 pairs of non-incident edges. Therefore, an asymptotic upper bound of the number of triangles and 4-cliques in $G$ gives an upper bound for $|E_{VCC}|$.

In any triangle, some vertex must come first in a degeneracy ordering, and can be in a triangle with at most $\binom{d}{2}$ of its at most $d$ later neighbors. Therefore each vertex is in $O(d^2)$ triangles with its later neighbors and, summing up over all vertices, contributes at most $O(d^2 n)$ edges to $E_{VCC}$. Similarly, for each edge $\{u, v\}$ we count the number of 4-cliques it is in with (non-incident) edges that come lexicographically after it in the degeneracy ordering. The number of triangles the second vertex can be in with later neighbors is $\binom{d}{2}$ and hence the edge is in at most $O(d^2)$ 4-cliques with $v$'s at most $d$ later neighbors, giving at most $O(d^2 m)$ 4-cliques total. Thus, we conclude that $|E_{VCC}| = O(d^2 m)$. ◀

Thus, $G_{VCC}$ has at size at most $O(d^2 m)$, a factor $O(d^2)$ larger than $G$. As a consequence, the average degree of the graph may increase, but by no more than a factor $O(d)$: whereas $G$ has average degree $2|E|/|V| = O(dn)/n = O(d)$, graph $G_{VCC}$ has average degree $2|E_{VCC}|/|V_{VCC}| = O(d^2 m)/m = O(d^2)$. Therefore, for input graphs with small degeneracy, the transformed graph is expected to be sparse as well.

However, even if the degeneracy $d$ is small, the graph $G_{VCC}$ may be very large in practice. Hence, to use this transformation, we require techniques to keep the graph size manageable.

## 5 Synergistic Reductions: Applying ECC and VCC Reductions

We propose to handle the blow-up by Kou et al. [25] by applying both ECC and VCC reductions to the problem, which we call *synergistic* data reduction. We first show how to adjust the transformation to work on reduced ECC instances, after which we can apply VCC reductions. We also explore the possibility of "lifting" VCC reductions to ECC reductions.

### 5.1 Transforming a Partially-Covered ECC Problem Kernel

Recall that the data reductions from Gramm et al. [19] result in a graph in which some edges are covered, which is not supported by the transformation of Kou et al. [25]. While it is tempting to modify the transformation to operate on *only* the uncovered edges $E'$, this does not necessarily result in an equivalent instance, as already-covered edges may still be needed to compute a minimum number of cliques covering $E'$. For instance, in Figure 1, covering edges $\{x, y\}$ and $\{y, z\}$ with the single clique $C_2$ uses the already-covered edge $\{x, z\}$.

One way to correct for this is to first perform the transformation on the entire graph $G = (V, E)$, and then take the subgraph induced by the vertices corresponding to uncovered edges in $E'$. However, this strategy is slow when the edge set $E$ is significantly larger than $E'$. We show that it is possible to perform the transformation without making vertices for all edges in $E$. Note that since all that remains is to cover the edges in $E'$, we now focus on covering all $E'$ using a minimum number of cliques in $G$. Taken together with already-chosen cliques from ECC reductions, this gives us a covering of all of $G$. (See Figure 3.)

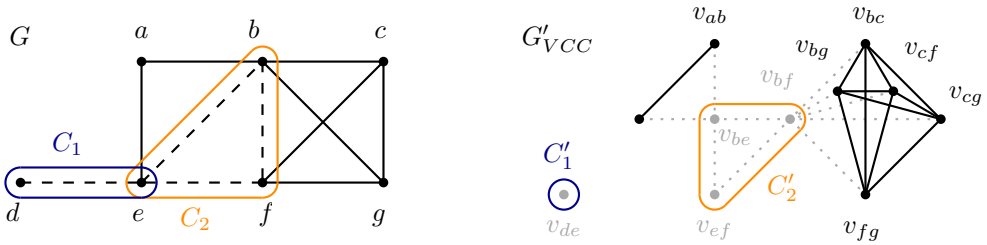



**Figure 3** A partially-covered graph $G$ with cliques $C_1$, $C_2$ already added to the cover, and its transformed graph $G'_{VCC}$. Grayed vertices and (dotted) edges are those in $G_{VCC}$, but not $G'_{VCC}$.

We transform $G$ to a graph $G'_{VCC} = (V'_{VCC}, E'_{VCC})$, where $V'_{VCC} = \{v_{xy} \mid \{x, y\} \in E'\}$ and $E'_{VCC} = \{\{v_{xy}, v_{wz}\} \mid \{x, y\}, \{w, z\} \in E'$ and $\{x, y\} \cup \{w, z\}$ is a clique in $G\}$. This transformation preserves cliques in $G$ that cover edges in $E'$, which we capture with the following observation.

▶ **Observation 3.** *If $C'$ is a clique in $G'_{VCC}$ then $C = \cup_{v_{xy} \in C'} \{x, y\}$ is a clique covering $|C'|$ edges of $E'$ in $G$.*

Furthermore, the transformation gives a correspondence between cliques covering $E'$ in $G$ and VCCs in $G'_{VCC}$.

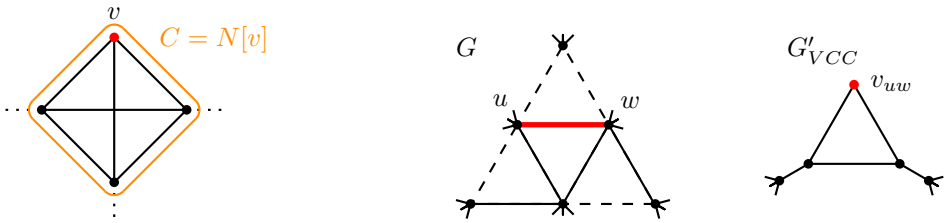
▶ **Theorem 4.** *If $\mathcal{C}'$ is a VCC in $G'_{VCC}$ then $\mathcal{C} = \{\cup_{v_{xy} \in C'} \{x, y\} \mid C' \in \mathcal{C}'\}$ is a set of cliques covering $E'$ in $G$.*

**Proof.** By Observation 3, every clique $C' \in \mathcal{C}'$ in $G'_{VCC}$ corresponds to a clique $C = \cup_{v_{xy} \in C'} \{x, y\}$ in $G$ that covers its corresponding edges of $E'$. Hence, a VCC that covers all $V'_{VCC}$ of $G'_{VCC}$ corresponds to a collection of cliques covering all edges $E'$ in $G$. ◀

Note that in Theorem 4, $|\mathcal{C}| = |\mathcal{C}'|$. Hence, a minimum VCC in $G'_{VCC}$ corresponds to a minimum-cardinality set of cliques covering $E'$ in $G$. This transformation gives us a technique for computing a minimum ECC: First apply the data reductions of Gramm et al., then compute $G'_{VCC}$ and use VCC reductions and any VCC solver to compute a minimum VCC in $G'_{VCC}$, giving us cliques covering $E'$ in $G$ and, ultimately an entire ECC of $G$. While applying VCC reductions to $G'_{VCC}$ may produce a smaller instance, these data reductions are not actually producing a smaller *ECC instance*. However, as we now show, we can also "lift" some VCC reductions to the ECC problem, by keeping the equivalence between cliques in the transformation in mind.

## 5.2 Lifting VCC Reduction Rules to ECC

Unlike the ECC problem, the VCC problem has many data reduction rules [31]. These include reductions based on simplicial vertices, dominance, twins, degree-2 folding, and crowns. We briefly discuss two classes of VCC reductions: clique-removal-based rules and folding-based rules. We place them in the context of the ECC problem, discuss whether it is viable to "lift" them to the ECC problem, and consider if the graph transformation is needed. By combining existing ECC reductions with VCC reductions, we aim to reduce ECC instances even further.




**(a)** $C$ is in a minimum VCC.

**(b)** Edge $\{u, w\} \in E'$ is in two cliques in $G$, but $v_{uw}$ in $G'_{VCC}$ is simplicial.

**Figure 4** The simplicial vertex VCC reduction can be applied after transforming $G$ to $G'_{VCC}$.

### 5.2.1 Clique-Removal-Based VCC Reductions

We call a VCC reduction that removes a set of cliques from the graph a *clique-removal-based* rule. Four VCC reductions (simplicial vertex, dominance, twin removal, and crown) are clique-removal-based rules [31]. Such rules can be easily transformed into an ECC reduction: By the equivalence between cliques in the problem transformation, stated in Observation 3, removing a clique in $G'_{VCC}$ is equivalent to covering its corresponding clique in $G$. Thus, to apply clique-removal-based VCC reductions directly to the ECC problem, we can compute $G'_{VCC}$, apply any clique-removal-based rules, and then cover these cliques in $G$. We capture this with the following theorem.

▶ **Theorem 5.** *Any clique-removal-based VCC reduction can be lifted to an ECC reduction.*

Of course, we could try to apply these reductions more efficiently to $G$ directly. We discuss two clique-removal-based VCC reductions and discuss whether they are worth implementing for ECC directly, or if we should transform $G$ to $G'_{VCC}$ first.

#### Simplicial Vertex Reduction

A vertex $v$ is *simplicial* if $N[v]$ forms a clique. In this case, the clique $C = N[v]$ is in some minimum VCC. (See Figure 4a.)

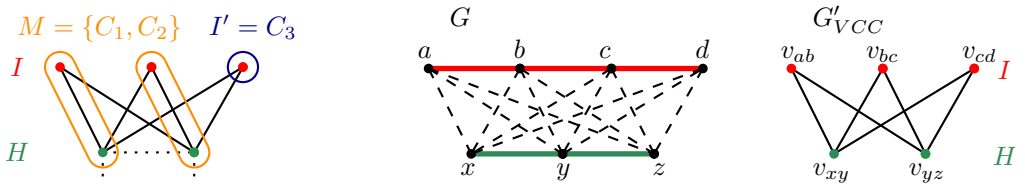
▶ **VCC Reduction 1** (Simplicial Vertex Reduction [31]). *Let $v \in V$ be a simplicial vertex. Then $C = N[v]$ is a clique in some minimum VCC. Add $C$ to the clique cover and remove $C$ from the graph.*

Applying VCC Reduction 1 on $G'_{VCC}$ is reminiscent of applying ECC Reduction 2 on the untransformed graph $G$. While it is true that for $\{u, w\} \in E'$, if $N_{\{u,w\}}$ is a clique in $G$, then $v_{uw}$ is simplicial in $G'_{VCC}$, the converse is not true in general. Hence, VCC Reduction 1 is more powerful. Consider the counterexample in Figure 4b. Vertex $v_{uw}$ is simplicial in $G'_{VCC}$, but $\{u, w\} \in E'$ is in two cliques of $G$.

Thus, we have a new data reduction for the ECC problem, which subsumes ECC Reduction 2:

▶ **ECC Reduction 5** (Lifted Simplicial Vertex Reduction). *Let edge $\{u, w\} \in E'$ and let set $C = \{x, y \in V \mid \{x, y\} \in E'$ and $\{u, w\} \cup \{x, y\}$ is a clique in $G\}$ be the set of vertices of edges in some clique with $\{u, w\}$. If $C$ is a clique, then add $C$ to the clique cover, and cover any uncovered edges of $C$ in $G$.*

To apply our lifted reduction, we could of course first compute $G'_{VCC}$ and then apply VCC Reduction 1. However, we can also apply it directly to $G$ with a slight modification to ECC Reduction 2. For each edge $\{u, w\} \in E'$ compute the common neighborhood $N_{\{u,w\}}$. Instead of checking that the common neighborhood is a clique, collect the uncovered edges




**(a)** A flared crown. Cliques $C_1$, $C_2$, and $C_3$ are in some minimum VCC.

**(b)** A partially-covered $G$ where $G'_{VCC}$ is a flared crown.

**Figure 5** The crown removal VCC reduction can be applied after transforming $G$ to $G'_{VCC}$.

between vertices in $N_{\{u,w\}}$, and check if they induce a clique. Since $|N_{\{u,w\}}| \leq \Delta$, it takes $O(d\Delta)$ to collect uncovered edges by iterating through the at most $d$ later neighbors of each vertex, which dominates the running time of this step. Exhaustively applying the reduction to all edges takes time $O(d\Delta m)$, which is slightly slower than the $O(d^2 m)$ time for ECC Reduction 2.

Is it worth applying ECC Reduction 5 directly to $G$, or should we first transform $G$ and run VCC Reduction 1 instead? The transformation can be done in time $O(d^2 m)$ by enumerating all of the triangles and 4-cliques of $G$ [11], hence performing the transformation is faster in theory than applying ECC Reduction 5 to $G$ directly. However, in $G'_{VCC}$ the largest clique may have as many as $\Theta(d^2)$ vertices and $\Theta(d^4)$ edges since a clique of size $d+1$ in $G$ has $\Theta(d^2)$ edges in $G$. Therefore, the time to apply VCC Reduction 1 for each of the $m$ vertices of $G'_{VCC}$ is $O(d^4 m)$. Thus, in theory, it is more efficient to apply ECC Reduction 5 directly, rather than first applying a conversion.

However, there are compelling reasons to perform the conversion. For one, most implementations of simplicial vertex reductions limit the degree of the vertex considered – in some cases to as small as two – since large-degree simplicial vertices rarely appear in sparse graphs. Therefore, in practice, it is unlikely that we would observe this large running time. However, a more compelling reason to perform the transformation is that there are two highly effective VCC reductions that we do not currently know how to apply directly to $G$. The first is the crown removal reduction (a clique-removal-based reduction) and the second is the degree-2 folding-based reduction.

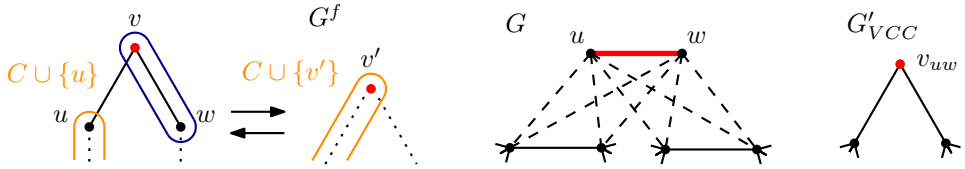
### Crown Removal Reduction

The crown removal reduction is arguably one of the most powerful data reductions, successfully reducing sparse instances for the minimum vertex cover and VCC problems [3, 4, 10].

In a pair of vertex sets $(H, I)$, $H$ is called a *head* and $I$ a *crown* if: $I$ is an independent set, $N(I) = H$, and there exists a matching from $H$ to $I$ of size $|H|$. Figure 5a shows a crown structure. Note that, due to the matching requirement, $|I| \geq |H|$. If $|I| = |H|$, the crown is called *straight*, otherwise it is *flared*. Strash and Thompson [31] give the following data reduction for the VCC problem, adapting a data reduction for the dual coloring problem [17].

▶ **VCC Reduction 2** (Crown Removal Reduction [31]). *Let $(H, I)$ be a head and crown with matching $M$ and unmatched vertices $I' \subseteq I$. Then add cliques in $M$ and $I'$ to the clique cover and remove $N[I]$ from the graph. (See Figure 5a.)*

Note that it is possible to identify flared crowns by applying a reduction based on an LP relaxation, originally introduced for the minimum vertex cover problem by Nemhauser and Trotter [29]. A variant of this algorithm due to Iwata et al. [21] identifies and removes *all* flared crowns at once by computing a maximum matching on a bipartite graph with $2n$ vertices and $2m$ edges using the Hopcroft-Karp algorithm [20] with running time $O(m\sqrt{n})$.




**(a)** The degree-2 folding VCC reduction.

**(b)** Edge $\{u, w\} \in E'$ in $G$ transforms into a degree-2 vertex $v_{uw}$ with non-adjacent neighbors in $G'_{VCC}$.

**Figure 6** The degree-2 folding VCC reduction can be applied after transforming $G$ to $G'_{VCC}$.

As Figure 5b illustrates, after exhaustively applying Gramm et al.'s [19] ECC reductions it is possible to have a crown structure after transforming to $G'_{VCC}$. Thus, lifting the crown removal reduction can further reduce an ECC instance. However, algorithms for computing a maximum matching for the LP relaxation use an explicit representation of $G'_{VCC}$ and therefore it is unclear how to run this reduction without first transforming $G$ to $G'_{VCC}$. The transformation and maximum matching can be computed in time $O(d^2m + d^2m\sqrt{m}) = O(d^2m^{3/2})$, since there are $O(m)$ vertices and $O(d^2m)$ edges in $G'_{VCC}$. We leave the question of whether the LP relaxation reduction can be more efficiently lifted to an ECC reduction as an open problem.

### 5.2.2 Folding-Based VCC Reductions

In contrast to clique-removal-based reductions, *folding-based* reductions contract a subset $S \subseteq V$ of vertices into a single vertex $v'$. *Folding* $S$ produces a new graph $G^f = (V^f, E^f)$ with $V^f = (V \setminus S) \cup \{v'\}$ and $E^f = (E \setminus \{\{v, x\} \in E \mid v \in S\}) \cup \{\{v', x\} \mid \exists v \in S, x \notin S, \{v, x\} \in E\}$. We discuss the connections between the ECC problem and the simplest folding-based reduction, folding vertices of degree two.

#### Degree-2 Folding

The degree-2 folding reduction for VCC contracts a degree-2 vertex $v$ with non-adjacent neighbors $u$ and $w$ that are *crossing independent* [31]. That is, for each edge $\{x, y\} \subseteq N(u) \cup N(w)$ either $\{x, y\} \subseteq N(u)$ or $\{x, y\} \subseteq N(w)$. This condition ensures that no spurious cliques are formed after folding. A vertex $v$ meeting these conditions is *foldable*.

▶ **VCC Reduction 3** (Degree-2 Folding Reduction [31]). *Let $v \in V$ be a foldable degree-2 vertex with non-adjacent neighbors $N(v) = \{u, w\}$. Let $G^f$ be the graph obtained by folding $\{v, u, w\}$. Let $\mathcal{C}^f$ be a minimum VCC of $G^f$ with clique $C_{v'} \in \mathcal{C}^f$ covering vertex $v'$ and let $C = C_{v'} \setminus \{v'\}$. Then, the clique cover*

$$\mathcal{C} = \begin{cases} (\mathcal{C}^f \setminus \{C_{v'}\}) \cup \{C \cup \{u\}, \{v, w\}\} & \text{if } C \subseteq N(u), \\ (\mathcal{C}^f \setminus \{C_{v'}\}) \cup \{C \cup \{w\}, \{v, u\}\} & \text{otherwise,} \end{cases}$$

*is a minimum VCC of $G$.*

See Figure 6a for an example of the degree-2 folding VCC reduction. We note that the transformation from an ECC instance to a VCC instance by Kou et al. [25] does not produce any degree-2 vertices with non-adjacent neighbors, as edges forming a triangle or 4-clique in $G$ form a triangle or 6-clique in $G_{VCC}$. However, our transformation with covered edges can result in such vertices (see Figure 6b). Thus, the degree-2 folding VCC reduction can be used to further reduce the instance when applied to $G'_{VCC}$.

We leave as an open problem whether folding-based rules can be lifted to new ECC reductions; we conjecture that it is possible to lift at least degree-2 folding. However, given how effective the degree-2 folding reduction is in practice for the VCC problem, we highly recommend applying it, even though it incurs the overhead of the transformation to $G'_{VCC}$.

### 5.3 Wrapping It All Up

With the tools in this section in hand, we have a clear path to solving the ECC problem on sparse graphs: first apply the data reductions due to Gramm et al. [19], then transform the partially-covered graph into a VCC instance, then apply VCC reductions and solve what remains with any VCC solver. We next perform experiments to evaluate this method.

## 6 Experimental Evaluation

We now compare our technique to the state of the art through extensive experiments on both synthetic instances and real-world graphs.

### 6.1 Experimental Setup

We implemented the ECC reductions and ECC to VCC transformation in C++ and integrated our methods with the VCC reductions and VCC algorithms by Strash and Thompson[2] [31], which we then compiled with `g++` version 11 using the `-O3` optimization flag. Our source code is available under the open source MIT license[3]. All experiments were conducted on Hamilton College's High Performance Computing Cluster (HPCC), on a machine running CentOS Linux 7.8.2003, with four Intel Xeon Gold 6248 processors running at 2.50GHz with 20 cores each, and 1.5TB of memory. Each algorithm is run sequentially on its own core.

We run experiments on six different algorithms. Gramm is the original branch-and-reduce code by Gramm et al. [19] written in OCaml, which we compiled with `ocamlc` version 3.10.2, and provided a sufficiently large stack size due to its heavy use of recursion. We implement three algorithms in C++ that first exhaustively apply ECC Reductions 1 and 2, perform a problem reduction to a VCC instance, apply VCC reductions, and then run a VCC solver: Redu$^3$BnR solves with the VCC branch-and-reduce algorithm by Strash and Thompson [31], Redu$^3$IG solves with the VCC iterated greedy (IG) heuristic algorithm by Chalupa [9], and Redu$^3$ILP solves with an assignment-based ILP formulation [22, 28] for VCC and Gurobi version 9.5.1. Finally, the two heuristic algorithms Conte [12] and EO-ECC [1] are from their respective authors and are compiled with `javac` version 8 and `g++` version 11 with `-O3`, respectively. Unless stated otherwise, we run each algorithm with a 24-hour time limit. Our stated running times do not include I/O time such as graph reading and writing.

In our tables, "Kernel" denotes the relevant size of the graph after reductions as either uncovered edges (Gramm) or vertices remaining (for VCC-based algorithms). "Time" is the time (in seconds) the solver takes to exactly solve the instance. A "–" indicates that the solver did not finish in the 24-hour time limit. **Bold** values indicate the value is the smallest among all algorithms in the table.

We run our experiments on randomly-generated instances as well as real-world graphs.

[2] `https://github.com/darrenstrash/ReduVCC`
[3] `https://github.com/darrenstrash/Redu3ECC`

**Erdős-Rényi Graphs.** We generate 70 instances of varying density using the $G(n, p)$ model of generating an $n$-vertex graph where each edge is selected independently with probability $p$. We use values of $n$ that are powers of two from 64 to 2048, with two different values of $p$ for each to show the effect of density on the tested algorithms. We generate 5 graphs with each $n$, $p$ pair using different random seeds to observe the behavior of algorithms on multiple instances of similar size and density.

**Real-World Instances.** We run our experiments on 52 large, sparse, complex networks from the Stanford Network Data Repository (SNAP)[4], the Laboratory for Web Algorithmics (LAW)[5], and the Koblenz Network Collection (KONECT)[6]. These graphs include citation networks, web-crawl graphs, and social networks; the largest graph has 18M vertices, and most graphs follow a scale-free degree distribution: there are many low degree vertices and few high degree vertices. The number of vertices and edges for each instance can be found with experimental results in Tables 2 and 3.

## 6.2 Results on Synthetic Instances

We begin by comparing the performance of Gramm and Redu$^3$BnR on synthetic instances generated with the Erdős-Rényi $G(n, p)$ model. In Table 1, we present the average kernel size and running time executing Gramm and Redu$^3$BnR on the 5 instances of each pair of $n$ and $p$. We disable ECC Reduction 3 in Gramm, since this configuration enables it to solve the largest number of synthetic instances within the time limit.

Focusing on running time, Gramm and Redu$^3$BnR are equally matched on very sparse graphs, quickly solving many instances in significantly less than one second. Though, as the density increases even slightly, which can be seen when fixing $n$ but increasing $p$, Gramm is no longer able to solve even small instances in a 24-hour time limit. However, on all instances, Redu$^3$BnR easily computes exact solutions. The reason why is clear: on instances that Gramm is unable to solve, Gramm's kernel is large (for the highest density instance with $n = 64, p = 0.2$, even a kernel of average size 100 is too large for Gramm to solve), whereas the VCC kernels for Redu$^3$BnR are significantly smaller in all cases. Indeed, for the densest graphs of each value of $n$, Gramm is unable to solve every instance in 24 hours, but Redu$^3$BnR solves all graphs in less than a second. This illustrates that the combined reduction power of ECC and VCC reductions is able to handle denser instances than running ECC reductions alone.

## 6.3 Solving Large Real-World Instances Exactly

We now see which graphs can be solved exactly by one of three algorithms: Gramm, Redu$^3$BnR, and Redu$^3$ILP. We disable ECC Reductions 3 and 4 in Gramm, since this configuration enables it to reduce all instances within the time limit. The results are presented in Table 2. Gramm was able to solve 12 of the 27 instances exactly; 10 of these graphs were solved because the kernel had 0 uncovered edges and the other two instances (`ca-CondMat` and `ca-GrQc`) had small kernels of less than 100 uncovered edges. However, Gramm exceeds the 24-hour time limit on the 15 other instances, even those with as few as 176 uncovered edges.

[4] `https://snap.stanford.edu/data/`
[5] `http://law.di.unimi.it/datasets.php`
[6] `http://konect.cc/`

**Table 1** Results on small Erdős-Rényi graphs of varying density. A "*" indicates that not all runs finished in the 24-hour time limit, "–" indicates that no runs finished in the 24-hour time limit.

| Graph | | | Gramm | | Redu$^3$BnR | |
|---|---|---|---|---|---|---|
| $n$ | $p$ | $m$ | Kernel | Time (s) | Kernel | Time (s) |
| 64 | 0.15 | 300 | 1 | < **0.01** | **0** | < **0.01** |
| 64 | 0.2 | 404 | 100 | 1 324.52* | **10** | < **0.01** |
| 128 | 0.1 | 805 | **0** | < **0.01** | **0** | < **0.01** |
| 128 | 0.15 | 1 218 | 491 | – | **51** | **0.03** |
| 256 | 0.075 | 2 433 | 42 | 0.02 | **0** | < **0.01** |
| 256 | 0.1 | 3 264 | 1 105 | – | **12** | **0.02** |
| 512 | 0.05 | 6 557 | 137 | 0.21 | **1** | **0.02** |
| 512 | 0.065 | 8 513 | 2 281 | – | **5** | **0.04** |
| 1 024 | 0.0365 | 19 072 | 1 259 | 153.21* | **4** | **0.08** |
| 1 024 | 0.0375 | 19 596 | 1 704 | – | **4** | **0.07** |
| 2 048 | 0.025 | 52 245 | 3 147 | 5.18 | **4** | **0.18** |
| 2 048 | 0.0275 | 57 488 | 7 235 | – | **5** | **0.20** |

In contrast, Redu$^3$BnR solves 18 of the instances. On all instances, the kernel computed by Redu$^3$BnR was smaller than that of Gramm, the smallest of which is on `zhishi-hudong-int`, which is reduced to 2% of the size of Gramm's kernel. With the exception of three instances (`email-EuAll`, `web-NotreDame`, and `web-Stanford`), every instance was reduced to at most 10% of Gramm's kernel size. However, the limitations of branch and reduce for the VCC problem begin to show on these instances. Similar to Gramm, Redu$^3$BnR only finishes within the 24-hour time limit on graphs with kernel size less than 100, and therefore its success is largely due to the reduction of the input instance (a pattern observed in other problems [30]). On the other hand, the Gurobi solver with an ILP formulation is able to solve kernels of much larger size, even up to 536 209 vertices (in the case of `eu-2005`).

## 6.4 Solving Remaining Instances Heuristically

We now look at the instances that could not be solved in the 24-hour time limit by any exact method. The results are presented in Table 3. Nine instances were reduced to VCC within the time limit of 24 hours, and the remaining instances were too large to finish in the time limit (not in the table). After fully transforming the input ECC instance to a reduced VCC instance, we ran the iterated greedy approach IG due to Chalupa et al. [9], which we call Redu$^3$IG, and compare its best solution with a lower bound from KaMIS, a state-of-the-art evolutionary algorithm for finding near-maximum independent sets on huge networks [26]. Four instances were solved to within 300 cliques of optimum, two of which (`soc-Slashdot0811` and `soc-Slashdot0902`) are within 100 cliques. The remaining instances are solved to within 6 000 cliques of optimum.

## 6.5 Summarizing the Quality of Existing Heuristic Solvers

Finally, using our exact results, we evaluate the quality of two heuristic solvers designed for large sparse graphs. We compare Conte, an algorithm by Conte et al. [12] designed for large sparse graphs, and EO-ECC by Abdullah et al. [1]. We run Conte and EO-ECC on all instances that were solved exactly (i.e., those from Table 2). The results are presented in Table 4.

**Table 2** Comparing exact algorithms Gramm, Redu$^3$BnR, and Redu$^3$ILP on real-world instances solved by at least one of the algorithms in a 24-hour time limit. Times marked with a "*" indicate that the algorithm's speed was due to programming language differences and not algorithmic improvements.

| Graph | | | Gramm | | Redu$^3$BnR | | Redu$^3$ILP |
|---|---|---|---|---|---|---|---|
| Name | $n$ | $m$ | Kernel | Time (s) | Kernel | Time (s) | Time (s) |
| ca-AstroPh | 18 772 | 198 050 | 2 837 | – | **0** | **0.33** | **0.33** |
| ca-CondMat | 23 133 | 93 439 | 62 | 1.74 | **0** | **0.10** | **0.10** |
| ca-GrQc | 5 242 | 14 484 | 9 | 0.15 | **0** | **0.02** | **0.02** |
| ca-HepPh | 12 008 | 118 489 | 491 | – | **0** | **0.16** | **0.16** |
| ca-HepTh | 9 877 | 25 973 | 176 | – | **0** | **0.03** | **0.03** |
| cnr-2000 | 325 557 | 2 738 969 | 755 617 | – | **23 880** | – | **10 727.29** |
| dblp-2010 | 326 186 | 807 700 | 868 | – | **0** | **1.98** | **1.98** |
| dblp-2011 | 986 324 | 3 353 618 | 8 898 | – | **50** | **9.13** | 9.88 |
| email-EuAll | 265 214 | 364 481 | 20 648 | – | **5 064** | – | **6.99** |
| eu-2005 | 862 664 | 16 138 468 | 5 555 826 | – | **536 209** | – | **12 966.59** |
| p2p-Gnutella04 | 10 876 | 39 994 | **0** | 0.34 | **0** | 0.05* | 0.05* |
| p2p-Gnutella05 | 8 846 | 31 839 | **0** | 0.23 | **0** | 0.05* | 0.05* |
| p2p-Gnutella06 | 8 717 | 31 525 | **0** | 0.33 | **0** | 0.04* | 0.04* |
| p2p-Gnutella08 | 6 301 | 20 777 | 261 | – | **17** | **0.04** | **0.06** |
| p2p-Gnutella09 | 8 114 | 26 013 | 214 | – | **5** | **0.04** | **0.08** |
| p2p-Gnutella24 | 26 518 | 65 369 | **0** | 0.91 | **0** | 0.10* | 0.10* |
| p2p-Gnutella25 | 22 687 | 54 705 | **0** | 0.63 | **0** | 0.08* | 0.08* |
| p2p-Gnutella30 | 36 682 | 88 328 | **0** | 1.27 | **0** | 0.09* | 0.09* |
| p2p-Gnutella31 | 62 586 | 147 892 | **0** | 2.14 | **0** | 0.23* | 0.23* |
| roadNet-CA | 1 965 206 | 2 766 607 | **0** | 115.17 | **0** | 5.60* | 5.60* |
| roadNet-PA | 1 088 092 | 1 541 898 | **0** | 45.75 | **0** | 2.94* | 2.94* |
| roadNet-TX | 1 379 917 | 1 921 660 | **0** | 73.21 | **0** | 3.64* | 3.64* |
| web-BerkStan | 685 230 | 6 649 470 | 2 096 936 | – | **152 581** | – | **6 753.27** |
| web-Google | 875 713 | 4 322 051 | 266 455 | – | **16 440** | – | **35.58** |
| web-NotreDame | 325 729 | 1 090 108 | 98 861 | – | **14 553** | – | **20.10** |
| web-Stanford | 281 903 | 1 992 636 | 523 480 | – | **57 463** | – | **981.82** |
| zhishi-hudong-int | 1 984 484 | 14 428 382 | 1 175 068 | – | **26 536** | – | **568.26** |

**Table 3** Heuristic solutions for graphs that could not be solved exactly in 24 hours. "lb" is a lower bound on $\theta_E(G)$ from KaMIS, "ub" is the smallest clique cover computed by Redu$^3$IG, and "Time" is the time in seconds for Redu$^3$IG to reach this result.

| Graph | | | KaMIS | Redu$^3$IG | |
|---|---|---|---|---|---|
| Name | $n$ | $m$ | lb | ub | Time (s) |
| as-skitter | 1 696 415 | 11 095 298 | 5 843 072 | 5 847 591 | 20 848.17 |
| email-Enron | 36 692 | 183 831 | 42 141 | 42 207 | 2 201.00 |
| soc-Epinions1 | 75 879 | 405 740 | 185 544 | 186 384 | 18 064.79 |
| soc-pokec-relationships | 1 632 803 | 22 301 964 | 12 222 248 | 12 227 949 | 21 451.91 |
| soc-Slashdot0811 | 77 360 | 469 180 | 328 018 | 328 079 | 3 073.75 |
| soc-Slashdot0902 | 82 168 | 504 230 | 351 012 | 351 072 | 3 125.21 |
| wiki-Talk | 2 394 385 | 4 659 565 | 3 645 692 | 3 648 312 | 21 088.53 |
| wiki-Vote | 7 115 | 100 762 | 34 789 | 35 004 | 21 424.48 |
| zhishi-baidu-relatedpages | 415 641 | 2 374 044 | 1 372 941 | 1 373 912 | 9 989.00 |

From among the 27 graphs, Conte solves five instances exactly. A further nine instances are solved within 50 cliques of optimal, and eight additional graphs are solved within 2 000 of optimal. EO-ECC, on the other hand, solves eight instances exactly (a superset of Conte's five) and solves these faster than Conte. Furthermore, EO-ECC finds 14 smaller solutions faster than Conte (Conte only finds four smaller solutions faster). However, a distinct negative is EO-ECC's running time and solution quality on `cnr-2000`, `eu-2005`, and `web-BerkStan`, which is much worse than Conte. We conclude that Conte gives consistently fast results with reasonable solutions, and EO-ECC is sometimes very fast and accurate, and other times not.

**Table 4** Evaluation of the quality of heuristic solvers Conte and EO-ECC on all graphs with known edge clique cover number $\theta_E(G)$. "ub" is the solution found by the given algorithm, and "Time" is the algorithm's time in seconds. Values of "ub" marked in **bold** indicates the algorithm found an optimal solution, with its time in **bold** if it did so faster than its competitor. Values of "ub" in *italics* indicate that an algorithm found an ECC smaller than its competitor, with its time in *italics* if it did so faster than its competitor.

| Graph $G$ | | | | Conte | | EO-ECC | |
|---|---|---|---|---|---|---|---|
| Name | $n$ | $m$ | $\theta_E(G)$ | ub | Time (s) | ub | Time (s) |
| `ca-AstroPh` | 18 772 | 198 050 | 15 134 | 15 481 | 0.92 | *15 373* | *0.50* |
| `ca-CondMat` | 23 133 | 93 439 | 16 283 | 16 378 | 0.54 | *16 307* | *0.07* |
| `ca-GrQc` | 5 242 | 14 484 | 3 737 | 3 749 | 0.15 | *3 739* | *0.01* |
| `ca-HepPh` | 12 008 | 118 489 | 10 031 | 10 142 | 0.69 | *10 097* | *0.35* |
| `ca-HepTh` | 9 877 | 25 973 | 9 190 | 9 264 | 0.19 | *9 212* | *0.02* |
| `cnr-2000` | 325 557 | 2 738 969 | 752 118 | *756 905* | *14.92* | 763 365 | 2 820.97 |
| `dblp-2010` | 326 186 | 807 700 | 186 834 | 187 395 | 2.22 | *186 968* | *0.44* |
| `dblp-2011` | 986 324 | 3 353 618 | 707 773 | 713 219 | 13.56 | *709 156* | *3.48* |
| `email-EuAll` | 265 214 | 364 481 | 297 092 | *298 943* | 2.58 | 299 257 | 2.14 |
| `eu-2005` | 862 664 | 16 138 468 | 2 832 059 | 2 883 585 | 108.67 | 3 032 337 | 8 458.21 |
| `p2p-Gnutella04` | 10 876 | 39 994 | 38 491 | **38 491** | 0.29 | **38 491** | **0.04** |
| `p2p-Gnutella05` | 8 846 | 31 839 | 30 523 | 30 527 | 0.25 | *30 525* | *0.04* |
| `p2p-Gnutella06` | 8 717 | 31 525 | 30 322 | 30 327 | 0.26 | *30 324* | *0.04* |
| `p2p-Gnutella08` | 6 301 | 20 777 | 19 000 | 19 042 | 0.20 | *19 012* | *0.03* |
| `p2p-Gnutella09` | 8 114 | 26 013 | 24 117 | 24 150 | 0.24 | *24 133* | *0.03* |
| `p2p-Gnutella24` | 26 518 | 65 369 | 63 725 | 63 726 | 0.41 | **63 725** | **0.06** |
| `p2p-Gnutella25` | 22 687 | 54 705 | 53 367 | **53 367** | 0.33 | **53 367** | **0.05** |
| `p2p-Gnutella30` | 36 682 | 88 328 | 85 821 | **85 823** | 0.52 | **85 821** | **0.10** |
| `p2p-Gnutella31` | 62 586 | 147 892 | 144 478 | **144 478** | 0.83 | **144 478** | **0.15** |
| `roadNet-CA` | 1 965 206 | 2 766 607 | 2 537 936 | 2 537 945 | 17.90 | **2 537 936** | **1.02** |
| `roadNet-PA` | 1 088 092 | 1 541 898 | 1 413 370 | **1 413 370** | 10.62 | **1 413 370** | **0.69** |
| `roadNet-TX` | 1 379 917 | 1 921 660 | 1 763 295 | 1 763 298 | 13.48 | **1 763 295** | **0.89** |
| `web-BerkStan` | 685 230 | 6 649 470 | 1 834 074 | *1 850 605* | *54.34* | 1 903 872 | 2 089.25 |
| `web-Google` | 875 713 | 4 322 051 | 1 242 770 | 1 254 107 | 24.96 | *1 251 672* | 33.10 |
| `web-NotreDame` | 325 729 | 1 090 108 | 451 424 | 453 864 | 7.09 | *453 805* | 7.31 |
| `web-Stanford` | 281 903 | 1 992 636 | 562 417 | *570 958* | *16.85* | 591 957 | 326.92 |
| `zhishi-hudong-int` | 1 984 484 | 14 428 382 | 10 557 244 | 10 698 424 | 123.45 | *10 678 121* | 322.89 |
| Summary (**#optimal** / *#smaller and faster*) | | | | (**5** / *4*) | | (**8** / *14*) | |

## 7 Conclusion and Future Work

We introduced a technique to further reduce ECC problem instances via VCC data reductions, enabling us to solve large sparse real-world graphs that could not be solved before. Critical to this technique is the ability to transform reduced ECC instances to the VCC problem,

through a modification of the polynomial-time reduction of Kou et al. [25]. The combined reduction power of ECC and VCC reductions, which we call *synergistic* data reduction, produces significantly smaller kernels than ECC reductions alone. Of particular interest for future work is integrating data reduction rules with existing heuristic algorithms for the ECC problem, trying to implement a more efficient LP relaxation ECC reduction without a transformation, and to see if folding-based reductions can be lifted to the ECC problem.

## References


1 Wali M. Abdullah and Shahadat Hossain. A sparse matrix approach for covering large complex networks by cliques. In Derek Groen, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Computational Science - ICCS 2022 - 22nd International Conference, London, UK, June 21-23, 2022, Proceedings, Part III*, volume 13352 of *Lecture Notes in Computer Science*, pages 505–517. Springer, 2022. doi:10.1007/978-3-031-08757-8_43.

2 Wali M. Abdullah, Shahadat Hossain, and Muhammad. A. Khan. Covering large complex networks by cliques—A sparse matrix approach. In D. Marc Kilgour, Herb Kunze, Roman Makarov, Roderick Melnik, and Xu Wang, editors, *Recent Developments in Mathematical, Statistical and Computational Sciences*, pages 117–127. Springer, 2021. doi:10.1007/978-3-030-63591-6_11.

3 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theor. Comput. Syst.*, 41(3):411–430, 2007. doi:10.1007/s00224-007-1328-0.

4 Tokuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609, Part 1:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

5 Alper Atamtürk, George L. Nemhauser, and Martin W.P. Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121(1):40–55, 2000. doi:10.1016/S0377-2217(99)00015-6.

6 Mathieu Blanchette, Ethan Kim, and Adrian Vetta. Clique cover on sparse networks. In *2012 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 93–102. SIAM, 2012. doi:10.1137/1.9781611972924.10.

7 Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. A branch-and-bound algorithm for cluster editing. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2022.13.

8 Robert C. Brigham and Ronald D. Dutton. On clique covers and independence numbers of graphs. *Discrete Mathematics*, 44(2):139–144, 1983. doi:10.1016/0012-365X(83)90054-7.

9 David Chalupa. Construction of near-optimal vertex clique covering for real-world networks. *Computing and Informatics*, 34(6):1397–1417, 2015. URL: http://www.cai.sk/ojs/index.php/cai/article/view/1276.

10 Lijun Chang, Wei Li, and Wenjie Zhang. Computing a near-maximum independent set in linear time by reducing-peeling. *Proc. 2017 ACM International Conference on Management of Data (SIGMOD '17)*, pages 1181–1196, 2017. doi:10.1145/3035918.3035939.

11 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.

12 Alessio Conte, Roberto Grossi, and Andrea Marino. Large-scale clique cover of real-world networks. *Information and Computation*, 270:104464, 2020. doi:10.1016/j.ic.2019.104464.

13 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: New incompressibility results. *ACM Trans. Comput. Theory*, 6(2), May 2014. doi:10.1145/2594439.

14 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs in near-optimal time. *ACM J. Exp. Algorithmics*, 18, 2013. doi:10.1145/2543629.

15 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes: Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.

16 Damir Ferizovic, Demian Hespe, Sebastian Lamm, Matthias Mnich, Christian Schulz, and Darren Strash. Engineering kernelization for maximum cut. In *Proc. 2020 Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 27–41. SIAM, 2020. doi:10.1137/1.9781611976007.3.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

18 Serge Gaspers and Edward J. Lee. Faster graph coloring in polynomial space. In Yixin Cao and Jianer Chen, editors, *Proc. 23rd International Computing and Combinatorics Conference (COCOON 2017)*, volume 10392 of *LNCS*, pages 371–383. Springer, 2017. doi:10.1007/978-3-319-62389-4_31.

19 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *J. Exp. Algorithmics*, 13, February 2009. doi:10.1145/1412228.1412236.

20 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

21 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1749–1761. SIAM, 2014. URL: https://dl.acm.org/doi/10.5555/2634074.2634201.

22 Adalat Jabrayilov and Petra Mutzel. New integer linear programming models for the vertex coloring problem. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 640–652. Springer, 2018. doi:10.1007/978-3-319-77404-6_47.

23 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, pages 85–103. Springer US, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

24 Mikko Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion-exclusion. In *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 583–590, 2006. doi:10.1109/FOCS.2006.11.

25 Lawrence T. Kou, Larry J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM*, 21(2):135–139, February 1978. doi:10.1145/359340.359346.

26 Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, August 2017. doi:10.1007/s10732-017-9337-x.

27 Don R. Lick and Arthur T. White. $k$-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970. doi:10.4153/CJM-1970-125-1.

28 Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996. doi:10.1287/ijoc.8.4.344.

**29** George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8(1):232–248, 1975. doi:10.1007/BF01580444.

**30** Darren Strash. On the power of simple reductions for the maximum independent set problem. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics (COCOON'16)*, volume 9797 of *LNCS*, pages 345–356. Springer, 2016. doi:10.1007/978-3-319-42634-1_28.

**31** Darren Strash and Louise Thompson. Effective data reduction for the vertex clique cover problem. In Cynthia A. Phillips and Bettina Speckmann, editors, *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, pages 41–53. SIAM, 2022. doi:10.1137/1.9781611977042.4.

**32** Ahammed Ullah. Clique cover of graphs with bounded degeneracy. *CoRR*, abs/2108.09851, 2021. arXiv:2108.09851.

**33** Johan M.M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011. doi:10.1016/j.dam.2011.07.001.

**34** David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007. doi:10.4086/toc.2007.v003a006.

# Threshold Testing and Semi-Online Prophet Inequalities

**Martin Hoefer**
Institute of Computer Science, Goethe University Frankfurt, Germany

**Kevin Schewior**
Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark

## Abstract

We study *threshold testing*, an elementary probing model with the goal to choose a large value out of $n$ i.i.d. random variables. An algorithm can *test* each variable $X_i$ once for some threshold $t_i$, and the test returns binary feedback whether $X_i \geq t_i$ or not. Thresholds can be chosen adaptively or non-adaptively by the algorithm. Given the results for the tests of each variable, we then select the variable with highest conditional expectation. We compare the expected value obtained by the testing algorithm with expected maximum of the variables.

Threshold testing is a semi-online variant of the gambler's problem and prophet inequalities. Indeed, the optimal performance of *non-adaptive* algorithms for threshold testing is governed by the standard i.i.d. prophet inequality of approximately $0.745 + o(1)$ as $n \to \infty$. We show how *adaptive* algorithms can significantly improve upon this ratio. Our adaptive testing strategy guarantees a competitive ratio of at least $0.869 - o(1)$. Moreover, we show that there are distributions that admit only a constant ratio $c < 1$, even when $n \to \infty$. Finally, when each box can be tested multiple times (with $n$ tests in total), we design an algorithm that achieves a ratio of $1 - o(1)$.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Theory of computation → Online algorithms; Theory of computation → Markov decision processes

**Keywords and phrases** Prophet Inequalities, Testing, Stochastic Probing

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.62

**Related Version** *Full Version*: https://arxiv.org/abs/2307.01776

**Funding** *Martin Hoefer*: Supported by DFG Research Unit ADYN (411362735) and grant 514505843. *Kevin Schewior*: Supported by the Independent Research Fund Denmark, Natural Sciences, grant DFF-0135-00018B.

**Acknowledgements** The authors gratefully acknowledge discussions with Daniel Schmand and Luca von der Brelie.

## 1 Introduction

Consider an application process in which $n$ job candidates are interviewed sequentially one by one for a single position. For each candidate, we assume the qualification for the job can be expressed by an i.i.d. non-negative random variable $X_i$ with known distribution $F$. The goal is to maximize the expected value of the selected candidate. To which extent is the optimal achievable value harmed by the online arrival of the candidates? This is the classic *gambler's problem*, in which the loss in expected value is expressed by prophet inequalities [22, 26, 9]. More precisely, in this model one usually assumes (i) an interview fully reveals the realization of the respective variable, and (ii) the requirement of timely feedback forces the decision maker to irrevocably accept or reject the candidate upon seeing its realization. For i.i.d. variables, the best-possible prophet inequality states that a candidate $\sigma$ can be selected such that $\mathbb{E}[X_\sigma] \geq \beta \cdot \mathbb{E}[\max\{X_1, \ldots, X_n\}]$, where $\beta \approx 0.745$ [19, 10].


The gambler's problem has been extremely popular over the last decades, but assumptions (i) and (ii) are often unrealistic. Even after a long interview, an interviewer is usually not fully aware of the entire set of exact qualifications of a candidate. Moreover, in many selection processes a decision does not have to be taken instantaneously. In this paper, we examine the consequences of an arguably more realistic set of conditions. First, instead of (i), we assume that each candidate only partly reveals their realization in the form of a *single bit of information*. As we observe rather directly, this assumption is asymptotically equivalent to allowing to make a single threshold query to each candidate. Second, instead of (ii), we assume that the selection must be made only at the end of the process.

More formally, we consider the *threshold testing* model. We assume that (i′), instead of revelation of $X_i$, we perform a single *threshold test* with some arbitrary threshold $t_i$, and the feedback is binary ("positive" in case $X_i \geq t_i$ or "negative" otherwise), and (ii′) a candidate must be chosen only after completing *all* threshold tests. Again denoting the selected candidate by $\sigma$, we are interested in bounding the loss in expected value using an inequality of the form $\mathbb{E}[X_\sigma] \geq c \cdot \mathbb{E}[\max\{X_1, \ldots, X_n\}]$ for $c \in (0, 1)$. We call this a *semi-online prophet inequality*. A testing algorithm that satisfies it is called *c-competitive* and has a *competitive ratio* of $c$.

There are four possible models emerging from the different choices of (i) vs. (i′) and (ii) vs. (ii′). The remaining two models do not require substantial analytical efforts. Indeed, when we only replace (i) by (i′), the consequences are trivial: There is an optimal algorithm for the standard gambler's problem that uses threshold tests. Thus, the existing optimal algorithm and its guarantees continue to apply because the space of algorithms only shrinks when we require threshold tests. Also, only replacing (ii) by (ii′) implies a trivial problem – one can see and choose an option with maximum value at the end, a 1-competitive strategy. In contrast, the main contribution of this paper is to show that, with (i′) and (ii′) simultaneously, a mathematically interesting model arises.

Our results also imply a stark qualitative distinction to the standard model. It is well known that adaptivity, i.e., allowing decisions to depend on past observations, does not help for the standard gambler's problem. In our model, we observe rather easily that non-adaptive testing algorithms are unable to asymptotically improve upon the ratio of $\beta \approx 0.745$. Our main result is a set of adaptive algorithms that improves significantly upon this bound and achieves a ratio of approximately 0.869. To complement this result, we show that there are distributions that imply a non-trivial asymptotical upper bound on the ratio, i.e., there is no $(1 - o(1))$-competitive algorithm. We proceed to discuss our contributions in more detail.

## 1.1 Techniques and Contribution

Let $F$ be the cumulative distribution function of the variables. For most of the paper we assume (essentially w.l.o.g.) that $F$ is continuous. Our algorithms perform quantile testing, i.e., they use thresholds of the form $F^{-1}(1 - q)$ for $q \in (0, 1)$, oblivious of other properties of the distribution. It is straightforward to achieve a competitive ratio of $1 - 1/\mathsf{e} > 0.632$ by using threshold $t_i = F^{-1}(1 - 1/n)$ for all variables and then choosing any variable that has been tested positively (if any); see, e.g., [19]. The analysis of this strategy is asymptotically tight for each of the following two parametric distributions[1]:

**$F_A$:** For some small $\varepsilon > 0$, with probability $1/\sqrt{n}$ choose a value uniformly from $[1-\varepsilon, 1+\varepsilon]$, and 0 otherwise. As $\varepsilon \to 0$ and $n \to \infty$, the algorithm gets a positive test and therefore value 1 with probability $1 - 1/\mathsf{e}$ while $\mathbb{E}[\max\{X_1, \ldots, X_n\}] = 1$.

[1] Strictly speaking, $F_A$ and $F_B$ are not continuous. For a rigorous argument, one can resort to an arbitrarily close continuous approximation of the distributions to obtain the same result.

$F_B$: Choose the value 1 with probability $1/n^2$ and 0 otherwise. The algorithm obtains a value 1 with probability $(1-(1-1/n)^n)/n$ while $\max\{X_1,\dots,X_n\}=1$ with probability $1-(1-1/n^2)^n$. As $n\to\infty$, the ratio of both probabilities approaches $1-1/\mathsf{e}$.

To improve upon $1-1/\mathsf{e}$, an algorithm needs to test for *both* smaller and larger thresholds than $F^{-1}(1-1/n)$. Thresholds that are all larger than $F^{-1}(1-1/n)$ decrease the ratio for $F_A$; thresholds that are all smaller than $F^{-1}(1-1/n)$ decrease the ratio for $F_B$.

The class of algorithms we consider here is parameterized by $\alpha_1,\dots,\alpha_k\in(0,1)$ with $\alpha_1>\alpha_2>\dots>\alpha_k$. In the beginning, such an algorithm uses $F^{-1}(1-\alpha_1/n)$ as a threshold until it sees a positive test. Generally, after $i<k$ positive tests, it sets $F^{-1}(1-\alpha_{i+1}/n)$ as a threshold. After $k$ positive tests (i.e., on $F^{-1}(1-\alpha_1/n),\dots,F^{-1}(1-\alpha_k/n)$), the algorithm can make arbitrary tests. Indeed, we eventually choose $\alpha_1>1$ and $\alpha_2<1$.

In our analysis, we exactly calculate the asymptotic probability that the algorithm sees precisely $i$ positive tests. Note that these probabilities asymptotically determine the probability density function of $X_\sigma$, the chosen variable. It is a step function in quantile space: The probability of making precisely $i$ positive tests is spread uniformly over the interval $[1-\alpha_i/n,1]$ for all $i\in\{1,\dots,k\}$.

We compare this probability density function of $X_\sigma$ with that of $\max\{X_1,\dots,X_n\}$ by stochastic dominance, leading to a tight analysis of such algorithms. For fixed values of $\alpha_1,\dots,\alpha_k$, we can analyze the competitive ratio of the respective strategy by solving a piecewise convex optimization problem, where the $k+1$ pieces correspond to the $k+1$ steps of the step function. We numerically maximize the minimum of this function.

We execute this analysis in detail for $k\in\{2,3\}$. For $k=3$, we obtain a competitive ratio of approximately 0.869 by setting $\alpha_1\approx 2.035$, $\alpha_2\approx 0.506$, and $\alpha_3\approx 0.057$. Our numerical results for $k=4$ indicate only negligible improvement by further increasing $k$.

We complement this result by a constant upper bound on the competitive ratio, i.e., an impossibility of achieving a competitive ratio of $1-o(1)$. Intuitively, there is a trade-off inherent in every test: Testing for a smaller value yields a fallback option in case only one positive test is found at the end; testing for a larger value allows to differentiate between different variables when multiple of them have been tested positively. There are instances in which, irrespective of how the algorithm solves this trade-off, it loses a constant in the competitive ratio. For the proof we consider a distribution where values 1, 2, or 3 occur with probability $1/n$ each, and 0 otherwise. It is minimal in the sense that a competitive ratio of $1-o(1)$ is achievable for any distribution that uses only three values in the support, or whose parameters do not depend on $n$.

Finally, we establish that a competitive ratio of $1-o(1)$ *can* be achieved using $n$ tests when a single variable can be tested multiple times (recall that the realization of each variable is only drawn *once initially* from the distribution). The idea is to drop $o(n)$ variables from consideration and test the remaining ones with a threshold such that, with high probability, $\max\{X_1,\dots,X_n\}$ is larger than this threshold, but only $o(n)$ of these tests are positive. The additional $o(n)$ tests can then be used to find the maximum variable among those that have been tested positively.

## 1.2 Further Related Work

The original prophet inequality [22] states that there is a 1/2-competitive algorithm in the setting of independent random variables with arbitrary distributions. Initiated by the work of Hajiaghayi, Kleinberg, and Sandholm [18], prophet inequalities have seen a surge of interest in the TCS community over the past 15 years. This has, for instance, led to the development of the tight i.i.d. prophet inequality with competitive ratio approximately 0.745 [10] as

well as almost-tight random-order [11, 5] and free-order [2, 25, 29, 5] prophet inequalities. Optimal or near-optimal prophet inequalities can be recovered without knowledge of the distribution but with $O(n)$ samples [7, 31, 8, 6]. Several works considered multiple-choice prophet inequalities with combinatorial constraints, e.g., [3, 13, 21, 12]. We also refer to the (at this point slightly outdated) surveys of Lucier [26] as well as Correa et al. [9] for additional references.

We compare our work with the two works from the prophet-inequality literature that seem closest. Orthogonally to samples, Li, Wu, and Wu [24] considered a version of the unknown i.i.d. setting in which quantile queries to the distribution can be made *before* the sequence of variables arrives. They as well as Perez-Salazar, Singh, and Toriello [30] also used a limited number of (quantile-based) thresholds to achieve near-optimal i.i.d. prophet inequalities. We are not aware of any version of the single-choice prophet inequality with general i.i.d. distributions to which the impossibility of approximately 0.745 does not apply.

In stochastic probing (e.g., [4, 15, 1, 16, 17]), information is also revealed online according to known distributions. The standard models are, however, quite different from our model: The decision maker gets to choose which variables to probe, and each probe entirely reveals the realization of the variable at hand. Eventually, the decision maker can pick a (set of) variable(s), much like in our model. Comparing with an omniscient optimum (like in the prophet inequality) is, however, usually hopeless in this setting. Instead, one focusses on computing or approximating the strategy that maximizes the expected selected (total) value, a task that is straightforward for our model.

In these probing models, the adaptivity gap measures the worst-case multiplicative gap between the value of the best adaptive and that of the best non-adaptive strategy. Note that, while our result does imply a nontrivial adaptivity gap (i.e., larger than 1) for our problem, we are studying a different question as we compare both adaptive and non-adaptive strategies with an omniscient optimum.

We are aware of two works in the probing literature in which tests do not eradicate all uncertainty about the respective variable. Hoefer, Schmand, and Schewior [20] considered a stochastic-probing model in which the first test to a variable only reveals whether the realization is above or below the median of the distribution, and additional tests can be used to further narrow down the realization in the same way applied to the conditional distribution. Gupta et al. [14] generalized the related classic Pandora's box problem due to Weitzman [32] and considered the Markovian model. There, a set of Markov chains, which correspond to variables that can eventually be chosen, is given and, in each step, a probe can be used to advance one of the Markov chains.

Threshold tests have also been considered in the context of estimating (properties of) a probability distribution. For example, Paes Leme et al. [23] gave bounds on the sample complexity, i.e., required number of such tests, to estimate the approximately optimal reserve price for certain types of distributions. Meister and Nietert [27] as well as Okoroafor et al. [28] investigated the sample complexity of estimating other objects, e.g., mean, median, or even full CDF, of the *empirical* distribution in a non-stochastic setting.

## 2 Preliminaries

We consider *threshold testing* defined as follows. We are given a distribution $F$ on $\mathbb{R}_{\geq 0}$ with finite expectation. There are $n$ *boxes*. Each box $i$ contains a hidden realization $X_1, \dots, X_n$ drawn *once upfront* i.i.d. from $F$. A *testing algorithm* can apply a *threshold test* to each box $i \in [n] = \{1, \dots, n\}$ exactly once, in that order. To apply a test to box $i$, the algorithm

chooses a threshold $t_i \geq 0$ and receives a binary feedback whether $X_i \geq t_i$ or not. Upon testing $i$, the algorithm learns if $X_i \geq t_i$ or not, but *not the precise value of* $X_i$. If $X_i \geq t_i$ we say the test was *positive*, otherwise it was *negative*. The algorithm may choose thresholds adaptively based on the feedback from earlier tests. Finally, after testing each box, the algorithm chooses one box $\sigma \in [n]$ and receives a reward of $X_\sigma$. Here, $\sigma$ is a random variable based on the observed feedback and the internal randomization of the algorithm. We call an algorithm $c$-competitive if $\mathbb{E}[X_\sigma] \geq c \cdot \mathbb{E}[\max\{X_1, \ldots, X_n\}]$. We are interested in maxmimizing $c$ in the limit as $n \to \infty$.

**Non-adaptive Algorithms and Prophet Inequalities.** Our testing problem has inherent connections to the classic prophet inequality for i.i.d. random variables. Consider the *non-adaptive* variant, in which the algorithm chooses thresholds $t_1, \ldots, t_n$ upfront. We observe that this problem is essentially the standard gambler's problem governed by prophet inequalities. The optimal algorithm for the gambler's problem emerges from straightforward backwards induction. For each box $i \in [n]$, the gambler sets a threshold $t_i$ to the expected profit from the optimal algorithm for boxes $i+1, \ldots, n$. The algorithm accepts $i$ if and only if $X_i \geq t_i$. It is straightforward to verify that this implies $t_1 \geq \ldots \geq t_n$. All $t_i$-values can be computed in advance. As such, a non-adaptive algorithm for threshold testing can use these thresholds and imitate the optimal algorithm for the gambler's problem.

▶ **Observation 1.** *The optimal non-adaptive testing algorithm for $n$ boxes obtains at least the expected reward of the optimal algorithm for the gambler's problem with $n$ boxes.*

We also observe the converse – for large $n$, the optimal reward of non-adaptive threshold testing is essentially the optimal reward in the gambler's problem.

▶ **Proposition 2.** *The optimal non-adaptive testing algorithm for $n$ boxes obtains at most the expected reward of the optimal algorithm for the gambler's problem with $n+1$ boxes.*

**Proof.** Consider the optimal non-adaptive algorithm for threshold testing. W.l.o.g. we can assume that the chosen thresholds are ordered such that $t_1^* \geq \ldots \geq t_n^*$. If at least one test is positive, then among the positively tested boxes, the algorithm chooses the one with the highest threshold – which is the earliest one in the sequence. The gambler can imitate this in the online model by using thresholds $t_1^*, \ldots, t_n^*$ and accepting the first one with $X_i \geq t_i^*$. If all tests are negative, then the testing algorithm accepts $X_1$ – it failed the test with the highest threshold and, as such, has the highest conditional expectation. Clearly, this is less than the apriori expectation of $F$, which can be obtained by the gambler from accepting box $X_{n+1}$. Hence, the gambler with $n+1$ boxes obtains more expected value. ◀

For large $n$ the best competitive ratio is approximately 0.745 by the optimal prophet inequality [19, 10]. For the rest of the paper we focus on *adaptive* testing algorithms.

**Threshold Testing vs. General Binary Feedback.** We discuss our scenario in the context of a more general model. In binary-feedback testing, the algorithm can choose a set $Y_i \subset \mathbb{R}_{\geq 0}$ and learns whether or not $X_i \in Y_i$. Note this model generalizes threshold testing – setting a threshold $t_i$ can be simulated by choosing $Y_i = \{x \in \mathbb{R} \mid x \geq t_i\}$. Nevertheless, the competitive ratio achievable is asymptotically the same as for threshold testing. As such, we restrict attention to threshold testing.

▶ **Proposition 3.** *The optimal algorithm for binary-feedback testing with $n$ boxes obtains at most the expected reward of the optimal algorithm for threshold testing with $n+1$ boxes.*

**Table 1** Numerically optimized parameters and competitive ratios for different values of $k$.

| $k$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ | comp. ratio as $n \to \infty$ |
|---|---|---|---|---|---|
| 1 | 1 | – | – | – | $1 - 1/e \approx 0.63212$ |
| 2 | 1.83298 | 0.35932 | – | – | $> 0.84005$ |
| 3 | 2.035135 | 0.5063 | 0.05701 | – | $> 0.86933$ |
| 4 | 2.038 | 0.508 | 0.058 | 0.0002 | $> 0.86956$ |

**Proof.** Consider an optimal algorithm for binary-feedback testing with $n$ boxes. We modify this algorithm to obtain an algorithm for threshold testing with $n+1$ boxes. We assume w.l.o.g. that, whenever the original algorithm chooses a set $Y_i$ to test the $i$-th box, then $\mathbb{E}[X_i \mid X_i \in Y_i] \geq \mathbb{E}[X_i]$ and, therefore, $\mathbb{E}[X_i \mid X_i \notin Y_i] \leq \mathbb{E}[X_i]$. We replace any such test with a threshold test for a threshold $t_i$ such that $\Pr[X_i \geq t_i] = \Pr[X_i \in Y_i]$, i.e., both tests are positive with precisely the same probability and $\mathbb{E}[X_i \mid X_i \geq t_i] \geq \mathbb{E}[X_i \mid X_i \in Y_i]$. We continue in the same way as the original algorithm would upon a positive or negative test, modifying subsequent tests in the same way. If the original algorithm eventually picks a box $i^\star$ with a positive test result, the new algorithm picks the same box. Thereby it obtain at least the same value since, by our choice of $t_{i^\star}$, $\mathbb{E}[X_{i^\star} \mid X_{i^\star} \geq t_{i^\star}] \geq \mathbb{E}[X_{i^\star} \mid X_{i^\star} \in Y_{i^\star}]$. Similarly, if the original algorithm would pick a box with a negative result, the new algorithm picks box $n+1$, obtaining $\mathbb{E}[X_{n+1}] = \mathbb{E}[X_{i^\star}] \geq \mathbb{E}[X_{i^\star} \mid X_{i^\star} \notin Y_{i^\star}]$ by our assumption above. ◀

## 3 Adaptive Testing

In this section, we prove the following theorem. For simplicity, we consider a continuous distribution $F$ throughout the proof. In the following section, we discuss that the result also generalizes to finite discrete distributions.

▶ **Theorem 4.** *There is an efficient* $(0.869 - o(1))$*-competitive algorithm for threshold testing with a continuous distribution.*

**Proof.** We consider a class of algorithms that is parameterized by a monotone sequence of quantile parameters $q_1, \ldots, q_k \in (0,1)$ where $q_1 > \ldots > q_k$. For convenience, we assume $q_0 = 1$ and $q_{k+1} = \ldots = q_n = 0$. The algorithm starts by testing for the $1 - q_1$ quantile of $F$. Since the distribution is continuous, $q_1$ corresponds to a threshold $\tau_1$ (i.e., $\tau_1$ is such that $Pr[X_i \geq \tau_1] = q_1$). Then for any $j \geq 1$, if the algorithm sees a negative test for $\tau_j$, it continues testing with $\tau_j$. If it sees a positive test for $\tau_j$, it increments $j$ to $j+1$ (i.e., continues testing with the next threshold $\tau_{j+1}$). After having tested each box, it selects the one with the best conditional expectation. This is either the box tested positively for the threshold corresponding to the largest quantile, or any box (when all tested negative for $\tau_1$).

We consider the values of $q_j$ in the form $q_j = \alpha_j/n$ for some $\alpha_j \in (0, n)$, for all $j \in [k]$. In Table 1, we give example values of $\alpha_j$ and the resulting competitive ratios for different values of $k$. We obtained these values by numerical optimization over a bounded interval.

We use $F$ to denote the CDF, i.e., $F(x) = \Pr[X_i < x]$, for each $i \in [n]$ and $x \in [0,1]$. For the maximum over $n$ i.i.d. draws, we obtain the CDF $F_m(x) = (F(x))^n = (\Pr[X_i < x])^n = \prod_i (\Pr[X_i < x]) = \Pr[\max_i X_i < x]$. We denote the complementary CDF by $G(x) = \Pr[X_i \geq x] = 1 - F(x)$ and $G_m(x) = \Pr[\max_i X_i \geq x] = 1 - F_m(x)$. Since $F$ is continuous, threshold $\tau_j = G^{-1}(q_j) = F^{-1}(1 - q_j)$, i.e., $G(\tau_j) = q_j$ and $F(\tau_j) = 1 - q_j$. Similarly, $F_m(\tau_j) = (1 - q_j)^n$ and $G_m(\tau_j) = 1 - (1 - q_j)^n$. We here restrict attention to values of $\alpha_j \in o(n)$, we will assume these are constants throughout. This implies $\lim_{n\to\infty} G_m(\tau_j) = 1 - \mathsf{e}^{-\alpha_j}$.

Our analysis proceeds via stochastic dominance. For any given threshold $t \geq 0$ we compare the complementary CDF $G_m(t)$ to the complementary CDF of our algorithm. We denote the latter by $A(t) = \Pr[X_\sigma \geq t]$, where $\sigma$ is the box chosen by our algorithm. If $A(t) \geq c \cdot G_m(t)$ for all $t \geq 0$, then the algorithm is $c$-competitive by stochastic dominance.

For any given $t \in [0, \infty)$ let $q = G(t) = 1 - F(t)$ and $\alpha = n \cdot q$. We will conduct our analysis with respect to $\alpha \in [0, n]$ instead of $t \in [0, \infty)$. We split $[0, n]$ into intervals $I_j = [\alpha_{j+1}, \alpha_j]$ for $j = 0, \ldots, k$, where we use $\alpha_0 = n$ and $\alpha_{k+1} = 0$. Suppose we see a positive test for $\alpha_j$. Then, between the positive test for $\alpha_{j-1}$ and the one for $\alpha_j$, assume there are $\ell_j \geq 0$ negative tests.

**Two Thresholds.** We start by discussing an algorithm with $k = 2$ thresholds. Suppose $\alpha \in I_2$. First, let's assume we only have a positive test for $t_1$ but not for $t_2$. We call this event $\mathcal{E}_{10}$. It happens with probability

$$\begin{aligned}\Pr[\mathcal{E}_{10}] &= \sum_{\ell_1=0}^{n-1} (1-q_1)^{\ell_1} q_1 \cdot (1-q_2)^{n-1-\ell_1} = q_1 \cdot (1-q_2)^{n-1} \cdot \frac{1 - \left(\frac{1-q_1}{1-q_2}\right)^n}{1 - \frac{1-q_1}{1-q_2}} \\ &= q_1 \cdot \frac{(1-q_2)^n - (1-q_1)^n}{q_1 - q_2} \quad = \quad \alpha_1 \cdot \frac{\left(1 - \frac{\alpha_2}{n}\right)^n - \left(1 - \frac{\alpha_1}{n}\right)^n}{\alpha_1 - \alpha_2} \ .\end{aligned}$$

In this case, the algorithm selects the box that was tested positive for $\tau_1$. It has a value at least $t$ with probability $q/q_1 = \alpha/\alpha_1$.

Otherwise, we have a positive test for $\tau_1$ and $\tau_2$, which we call event $\mathcal{E}_{11}$. The event that we have a positive test for $\tau_1$ (irrespective of what happens for $\tau_2$) is called $\mathcal{E}_1$. Clearly,

$$\Pr[\mathcal{E}_{11}] = \Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_{10}].$$

In case $\mathcal{E}_{11}$ happens, we select the box that tested positive for $\tau_2$. It has a value at least $t$ with probability $q/q_2 = \alpha/\alpha_2$.

Overall, for $\alpha \in I_2$ we see

$$\begin{aligned}A(\alpha) &= \frac{\alpha}{\alpha_1} \Pr[\mathcal{E}_{10}] + \frac{\alpha}{\alpha_2} \Pr[\mathcal{E}_{11}] = \alpha \cdot \left( \frac{\Pr[\mathcal{E}_{10}]}{\alpha_1} + \frac{\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_{10}]}{\alpha_2} \right) \\ &= \alpha \cdot \left( \frac{\Pr[\mathcal{E}_1]}{\alpha_2} - \frac{(\alpha_1 - \alpha_2) \Pr[\mathcal{E}_{10}]}{\alpha_1 \alpha_2} \right) = \frac{\alpha}{\alpha_2} \cdot \left(1 - \left(1 - \frac{\alpha_2}{n}\right)^n\right) \\ &= c_2(\alpha) \cdot \left(1 - \left(1 - \frac{\alpha}{n}\right)^n\right) = c_2(\alpha) \cdot G_m(\alpha) \ .\end{aligned}$$

Hence,

$$c_2(\alpha) = \frac{\alpha}{\alpha_2} \cdot \frac{1 - \left(1 - \frac{\alpha_2}{n}\right)^n}{1 - \left(1 - \frac{\alpha}{n}\right)^n} \geq \lim_{\alpha \to 0} c_2(\alpha) = \frac{1 - \left(1 - \frac{\alpha_2}{n}\right)}{\alpha_2} \geq \frac{1 - \mathsf{e}^{-\alpha_2}}{\alpha_2} \ ,$$

since for every given $n \geq 1$ and every $\alpha > 0$, the ratio $\alpha/(1 - (1 - \alpha/n)^n) > 1$, because $\alpha \geq 1 - (1 - \alpha/n)^n$ by concavity of the latter function.

Now for $\alpha \in I_1$, we consider the case with a positive test on $\tau_1$ but not on $\tau_2$. In this case, the box has a value of at least $t$ with probability $q/q_1 = \alpha/\alpha_1$. Alternatively, if we see a positive test for $\tau_1$ and $\tau_2$, the algorithms selects a box with a value of at least $t$ with probability 1. Overall, for $\alpha \in I_1$

$$A(\alpha) = \frac{\alpha}{\alpha_1} \cdot \Pr[\mathcal{E}_{10}] + \Pr[\mathcal{E}_{11}] \ = \ \Pr[\mathcal{E}_1] - \left( \frac{\alpha_1 - \alpha}{\alpha_1} \right) \cdot \Pr[\mathcal{E}_{10}]$$

$$= 1 - \left(1 - \frac{\alpha_1 - \alpha}{\alpha_1 - \alpha_2}\right)\left(1 - \frac{\alpha_1}{n}\right)^n - \frac{\alpha_1 - \alpha}{\alpha_1 - \alpha_2}\left(1 - \frac{\alpha_2}{n}\right)^n$$
$$= c_1(\alpha) \cdot \left(1 - \left(1 - \frac{\alpha}{n}\right)^n\right).$$

Since $\alpha \in [\alpha_2, \alpha_1]$ is a constant,

$$\lim_{n\to\infty} c_1(\alpha) = \frac{1}{1 - \mathsf{e}^{-\alpha}} \cdot \left(1 - \mathsf{e}^{-\alpha_1} - \frac{(\alpha_1 - \alpha)(\mathsf{e}^{-\alpha_2} - \mathsf{e}^{-\alpha_1})}{\alpha_1 - \alpha_2}\right).$$

Finally, for $\alpha \in I_0$, we see that

$$A(\alpha) = \frac{\alpha - \alpha_1}{n - \alpha_1}(1 - \Pr[\mathcal{E}_1]) + \Pr[\mathcal{E}_1] = \frac{\alpha - \alpha_1}{n - \alpha_1}\left(1 - \frac{\alpha_1}{n}\right)^n + \left(1 - \left(1 - \frac{\alpha_1}{n}\right)^n\right)$$
$$= c_0(\alpha) \cdot \left(1 - \left(1 - \frac{\alpha}{n}\right)^n\right).$$

Thus,

$$c_0(\alpha) \geq \frac{1 - \left(1 - \frac{\alpha_1}{n}\right)^n}{1 - \left(1 - \frac{\alpha}{n}\right)^n} \geq \frac{1 - \mathsf{e}^{-\alpha_1}}{1 - \mathsf{e}^{-\alpha}},$$

where the latter bound holds for any $n \geq 1$ and any constant $\alpha$. Indeed, when $\alpha \in \omega(1)$, we obtain a bound of $1 - \mathsf{e}^{-\alpha}$ in the limit for $n \to \infty$.

As a sanity check, observe that $c_1(\alpha_1) = c_0(\alpha_1) = 1$. Indeed, suppose we have a box with value $t \geq \tau_1$. Then either this box is tested positive for $\tau_1$, or some other box was tested positive for $\tau_1$ before. In either case, the algorithm indeed selects a box of value at least $\tau_1$. Similarly, observe that $c_2(\alpha_2) = 1$ as well. Indeed, suppose we have a box with value $t \geq \tau_2$. Suppose (1) this box is tested positive for $\tau_2$. Then it is selected. Suppose (2) the box is tested positive for $\tau_1$. Then it is selected, unless some later box is tested positive for $\tau_2$. Either way, we eventually obtain a value of at least $\tau_2$. Finally, suppose (3) the box is not tested at all. Then we have already selected a box of value at least $\tau_2$ before.

To obtain the best ratio, we strive to select constants $0 < \alpha_2 < \alpha_1$ in order to

$$\max_{\alpha_1,\alpha_2} \{\min_{\alpha\in I_2} c_2(\alpha), \min_{\alpha\in I_1} c_1(\alpha), \min_{\alpha\in I_0} c_0(\alpha)\}.$$

For $c_2(\alpha)$ and $c_0(\alpha)$ we obtain fairly clear lower bounds, which even hold pointwise for any $n$. It seems unpromising to obtain an insightful analytic formula for the minimum of $c_1(\alpha)$ as a function of $\alpha_1$ and $\alpha_2$. Instead, we numerically optimized parameters $\alpha_1, \alpha_2$ and used standard solver software to minimize $c_1(\alpha)$. The lower bounds for $c_2$ and $c_0$ then amount to $(1 - \mathsf{e}^{-0.35932})/0.35932 \geq 0.8400637\ldots$ and $1 - \mathsf{e}^{1.83298} \geq 0.8400564\ldots$. The minimum of $\lim_{n\to\infty} c_1(\alpha)$ is located roughly at $\alpha^* \approx 0.832961265\ldots$ with a value for $c_1(t) = 0.8400569\ldots$ For a plot of the ratios see Figure 1.

Along similar lines, we analyze the case with $k = 3$ thresholds in the full version, which yields a ratio of at least $0.869 - o(1)$ (see Table 1). Based on similar calculations, we also numerically optimized the case with $k = 4$, but we see only very slight improvements. Intuitively, the probability to reach a state with positive tests for all $k$ thresholds becomes extremely small. Increasing this probability requires to decrease the value to be tested for in the first $k - 1$ tests. However, the possibility to obtain an improvement in this way seems to vanish very quickly as $k$ grows larger. We conjecture that for all values of $k$, we cannot significantly improve the competitive ratio beyond 0.869 as $n \to \infty$. ◀

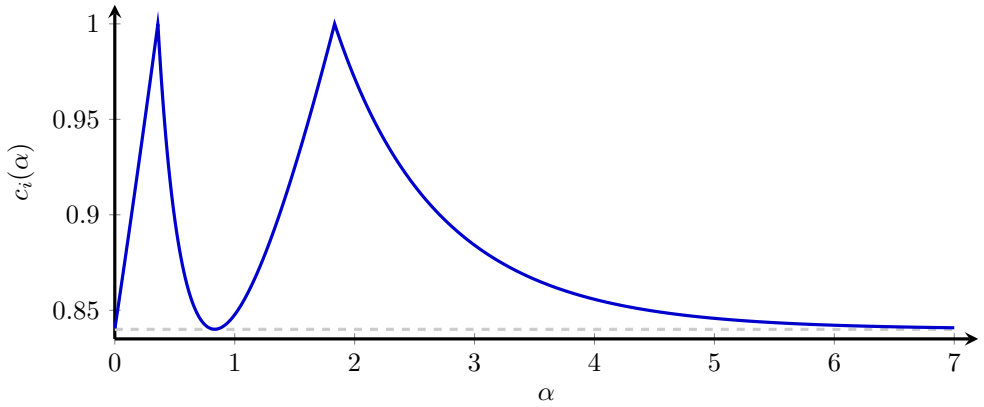



**Figure 1** Ratios $c_2$, $c_1$ and $c_0$ in the limit for $n \to \infty$ using $\alpha_1 = 1.83298$ and $\alpha_2 = 0.35932$.

Observe that the analysis of our algorithms is tight. Consider the value of $\alpha'$ that yields the minimum of all $c_i(\alpha)$ in the respective intervals $I_i$. For a "golden nugget"-distribution, where each $X_i$ has value 1 with probability $\alpha'/n$ and 0 otherwise, the above calculations become exact, and the analysis of the competitive ratio becomes tight. While, strictly speaking, this golden-nugget distribution is discrete, it is straightfoward to approximate it arbitrarily closely by a continuous distribution.

## 4 Discrete Distributions

Let us shift attention from a continuous distribution to a finite discrete distribution $F$. We assume $F$ is represented in straightforward form as a list of (value, probability) pairs. We denote by $m$ the number of distinct realizations, and we use $v_1 < v_2 < \ldots < v_m$ to denote the support of $F$.

Observe that w.l.o.g. we only need to test for these values $v_j$. If we test for a threshold $t$ in between two consecutive $v_j < t \leq v_{j+1}$, we obtain the same result by testing for $t = v_{j+1}$ instead. As such, we restrict to tests for values in the support.

### 4.1 Testing Algorithms

We first observe that an optimal testing algorithm can be computed in polynomial time. Moreover, we show that this algorithm yields a competitive ratio of $0.869 - o(1)$.

▶ **Theorem 5.** *For finite discrete distributions, an optimal testing algorithm can be computed by dynamic programming in polynomial time.*

**Proof.** We use backwards induction. Consider the last test of box $n$. Clearly, given the previously tested boxes $1, \ldots, n-1$, we can restrict attention to the one with the highest conditional expectation. We denote this value by $V^*_{n-1}$. Since each box is tested for exactly one of the $m$ realizations, there are $2m$ different possibilities for $V^*_{n-1}$. There are $m$ possible tests for box $n$. We can enumerate all the $2m^2$ combinations. For each value of $V^*_{n-1}$, the optimal test of box $n$ is the one leads to the highest expected value of the chosen item. Thus, to determine and describe the optimal decision for box $n$, we only need to consider $2m$ options of $V^*_{n-1}$, and for each option we record the best of the $m$ possible tests for box $n$.

For the induction, let $V^*_{i-1}$ and $V^*_i$ be the conditional expectation of the best tested box before and after testing box $i$, resp. Suppose that for each possible value of $V^*_i$, we have computed an optimal testing strategy for subsequent boxes $i+1, \ldots, n$, along with the

resulting expected value of $X_\sigma$. Now for box $i$, consider each of the $2m$ possible values for $V^*_{i-1}$. For each realization $v_k$, we can determine the effect when we test box $i$ for $v_k$ – i.e., the probability that $V^*_i = V^*_{i-1}$ (when the test on $i$ implies the conditional expectation of $i$ is at most $V^*_{i-1}$), as well as the probability that $V^*_i$ becomes any higher value (otherwise). For the resulting $V^*_i$, we inspect the value obtained by an optimal testing strategy for boxes $i+1, \dots, n$. This serves to find the test of box $i$ resulting in the optimal expected value.

Overall, to determine and describe the optimal decision for box $i$, we need to consider $2m$ options of $V^*_{i-1}$, and for each option we determine the best of the $m$ possible tests for box $i$ (using the results of the subsequent optimal testing strategy for boxes $i+1, \dots, n$). Finally, for box 1 $V^*_0$ is undefined. At this point, we only need to find the best of the $m$ possible tests for box 1 (using the results of the subsequent optimal testing strategy for boxes $2, \dots, n$). This concludes the backwards induction.

We record for each possible value $V^*_{i-1}$ the best threshold to test box $i$ along with the resulting expected value emerging from an optimal algorithm for boxes $i+1, \dots, n$. Hence, we can describe an optimal testing strategy using $2 \cdot (1 + (n-1) \cdot 2m))$ entries. This strategy can be computed in polynomial time via dynamic programming as described above. ◀

At this point, it is unclear how to apply our algorithm from the previous section to finite discrete distributions since $F^{-1}(1-q)$ may not be defined for the relevant values of $q$. In fact, we will show that the optimal algorithm in Theorem 5 achieves a competitive ratio of at least $0.869 - o(1)$ for finite discrete distributions.

We consider the following different model for testing discrete distributions, called *probability testing*. It can be viewed as the limit that emerges from approximating discrete with continuous distributions arbitrarily close. Here a test requires an input value $q \in [0,1]$. It then returns whether or not the value $v$ in the box lies in the top-$q$ fraction of probability mass of $F$. For a finite discrete distribution $F$, let $k$ be such that $\sum_{j=k+1}^{m} \Pr[v = v_j] < q \leq \sum_{j=k}^{m} \Pr[v = v_j]$. Then the test is positive for $v \in \{v_{k+1}, \dots, v_m\}$ and negative for $v \in \{v_1, \dots, v_{k-1}\}$. For $v = v_k$ it yields a randomized outcome, i.e., positive with probability $p_q = \left(q - \sum_{j=k+1}^{m} \Pr[v = v_j]\right) / \Pr[v = v_k]$ and negative otherwise. Hence, the overall probability that box $i$ is tested positive is exactly $q$.

Clearly, our algorithm from Section 3 can be implemented with probability testing and obtains a competitive ratio of $0.8969 - o(1)$. Probability and threshold testing are equivalent for continuous distributions, since there is a bijection between thresholds and values for $q$. For finite discrete distributions we observe in Proposition 6 that any algorithm for probability testing can be simulated using randomized threshold tests. We then show that randomized tests are not beneficial, i.e., for any algorithm with randomized threshold tests, there is one with *deterministic* tests performing at least as good. For a formal proof, see the full version.

▶ **Proposition 6.** *If there is a $c$-competitive algorithm for probability testing, then there is a $c$-competitive algorithm for threshold testing.*

▶ **Corollary 7.** *The optimal algorithm for finite discrete distributions is at least $(0.869 - o(1))$-competitive for threshold testing.*

## 4.2 Impossibility

Complementing our results in the previous subsection, we proceed to show a constant upper bound on the competitive ratio for $n \to \infty$.

▶ **Theorem 8.** *There exists no $(1 - o(1))$-competitive algorithm for threshold testing.*

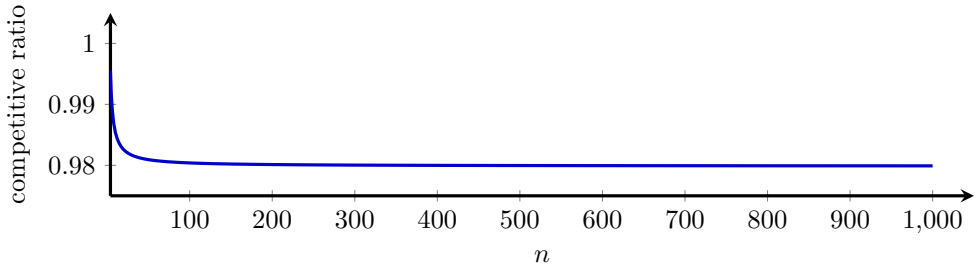



**Figure 2** The best-possible competitive ratio on the instance used in the proof of Theorem 8 as a function of $n$.

To prove the theorem, we are going to construct a counter example that is a discrete distribution, which carries over to the continuous case by the arguments given in Section 4. We first observe that such a distribution needs to depend on $n$: Otherwise, the top realization appears with constant probability in each box, and an algorithm simply testing for that realization finds it with probability $1 - o(1)$. Furthermore, such a distribution needs to have a support of cardinality at least 4: If the cardinality of the support is 3, it is w.l.o.g. exactly 3, and the algorithm can obtain $\max\{X_1, \dots, X_n\}$ by testing for the middle realization and, upon a positive test, testing for the top realization. If it finds a positive test on the top realization, it clearly obtains $\max\{X_1, \dots, X_n\}$ by choosing the corresponding box. If it finds a positive test on the middle realization, the corresponding box is the only one that can possibly contain the top realization, which the algorithm obtains by picking it, so it also obtains $\max\{X_1, \dots, X_n\}$. In the final case, $\max\{X_1, \dots, X_n\}$ is only the lowest realization, which the algorithm will also obtain by choosing any box.

We consider boxes that contain a realization 3, 2, or 1 with probability $1/n$ each and 0 otherwise. Intuitively, any algorithm that does not always test for the value 1 before encountering a positive test runs the risk of missing a value 1. Similarly, any algorithm that does not always test for the value 2 afterwards and before encountering another positive test runs the risk of missing a value 2. Such an algorithm, however, with constant probability, gets into a situation in which it has encountered precisely two positive tests, specifically, for the values 1 and 2. In that situation, it is clearly optimal to choose the box that has been positively tested for the value 2. With a constant probability, the value of this box is, however, equal to 2 while the one that has been tested positively for value 1 is equal to 3. The conclusion is that the algorithm, in any case, loses a constant fraction of $\mathbb{E}[\max\{X_1, \dots, X_n\}]$. In the full version, we present a formal version of this argument.

We have verified numerically (by solving the dynamic program from Theorem 5) that for this distribution the achievable competitive ratio decreases in $n$ in the interval $n = 2, \dots, 1000$. For $n = 1000$, the optimal competitive ratio is ca. 0.9799 (computed with full precision). See Figure 2 for the results.

## 5 Multiple Tests per Box

In this section, we consider a setting with $n$ boxes and a budget of $n$ threshold tests. Each box can be tested an arbitrary number of times with different thresholds[2] as long as there are still tests available. We again assume continuous distributions and show the following result.

[2] Recall that nature draws *initially a single* value $X_i \sim F$ inside each box $i$. All tests on the same box are evaluated accordingly. The results of multiple tests on the same box are all consistent with the single unknown $X_i$ drawn upfront.

▶ **Theorem 9.** *There is an efficient* $(1 - o(1))$*-competitive algorithm for threshold testing with multiple tests per box and a continuous distribution.*

**Proof.** In the first step, our algorithm discards the last $\lceil n^{2/3} \rceil$ boxes, losing only a $\lceil n^{2/3} \rceil / n$ fraction of the value. The remaining ones are tested for the threshold $F^{-1}(1 - n^{-1/3})$. Let $P$ be the set of boxes that were tested positively. For each box $i \in P$, the algorithm next searches the integers $\{0, \dots, \lfloor n^{2/3} \rfloor\}$ to find the largest $j$ such that the test for $F^{-1}(1 - n^{-1/3} + j/n)$ is negative. Then[3], $F^{-1}(1 - n^{-1/3} + j/n) < X_i \leq F^{-1}(1 - n^{-1/3} + (j+1)/n)$. Using a binary search, this requires at most $\lceil 2/3 \cdot \log n \rceil$ tests for each box $i \in P$. Using the result, we say that box $i$ is of type $j$. Since there are potentially up to $n$ boxes in $P$, the algorithm may well run out of tests during this process. Eventually, if the algorithm succeeds to determine the type of each box in $P$, it picks a box from $P$ with the highest type. If no such box exists, it is not unique, or the algorithm ran out of tests before determining the type of each box in $P$, it may choose an arbitrary box.

To analyze our algorithm, we fix any $v \in [F^{-1}(1 - n^{-1/3}), F^{-1}(1))$. We denote by $\mathcal{M}_v$ the event that $\max\{X_1, \dots, X_n\} = v$, and by $X_\sigma$ the value obtained by the algorithm. Our goal is to show that, whenever an optimal box has a high value $v$, the probability that we choose an optimal box is

$$\Pr[X_\sigma = v \mid \mathcal{M}_v] = 1 - o(1). \tag{1}$$

Now we see an optimal box with a high value with probability

$$\Pr[\max\{X_1, \dots, X_n\} \geq F^{-1}(1 - n^{-1/3})] = 1 - (1 - n^{-1/3})^n = 1 - o(1),$$

so proving Eq. (1) indeed suffices to prove the theorem. To show Eq. (1), we define two additional events:

- $\mathcal{E}_1$ is the event that $|P| \leq n^{1/2}$ (in particular, this implies that the algorithm does not run out of tests for large-enough $n$),
- $\mathcal{E}_2$ is the event that only a single box has the largest type.

Note that $\Pr[X_\sigma = v \mid \mathcal{M}_v] \geq \Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \mid \mathcal{M}_v]$ since our algorithm chooses the box with the maximum value if both $\mathcal{E}_1$ and $\mathcal{E}_2$ occur. We finalize the argument by observing that

$$\begin{aligned}\Pr[\mathcal{E}_1 \cap \mathcal{E}_2 \mid \mathcal{M}_v] &= \Pr[\mathcal{E}_1 \mid \mathcal{M}_v] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{M}_v \cap \mathcal{E}_1] \\ &\geq \left(1 - e^{-\frac{n^{1/3}}{3}}\right) \cdot \left(1 - \frac{n^{-2/3}}{1 - n^{-1/3}}\right)^{n^{1/2}} = 1 - o(1).\end{aligned}$$

For the inequality, we bound the first probability using a one-sided multiplicative Chernoff bound with $\mu = n^{1/3}$ and factor 2. We bound the second probability by observing that, conditioned on $\mathcal{M}_v$, the probability that a single box other than that with realization $v$ has the same type is at most $n^{-2/3}/(1 - n^{-1/3})$. Here, $n^{-2/3}$ is an upper bound on the probability of having the same type, and $1 - n^{-1/3}$ is a lower bound on the probability that an independently drawn value is below $v$ (using that $v$ is a high value). The additional condition on $\mathcal{E}_1$ does not increase the probability of $\mathcal{E}_2$. This shows Eq. (1) and thus completes the proof. ◀

It is rather straightforward to apply the insights from Sec. 4 to show similar results for testing with multiple tests per box and a finite discrete distribution. Our algorithm in the proof of Theorem 9 can be cast as a *sequential* testing algorithm: It tests boxes

[3] We use the convention $F^{-1}(x) = F^{-1}(1)$ for $x > 1$.

sequentially from box 1 to $n - \lceil n^{2/3} \rceil$. For each box $i$ it applies tests to determine whether $i \in P$ or not, and then binary search the type of $i$ (or aborts when it runs out of tests). For finite discrete distributions, we can optimize over such sequential testing algorithms using backwards induction, much like in the proof of Theorem 5. When considering box $i$, an optimal decision about the next test can be found by relying on three additional parameters. Apart from the best conditional expectation of a previous box $V^*_{i-1}$, we also consider the smallest realization for which we saw a positive test for $i$, the largest one for which we saw a negative test for $i$, as well as the number of tests we applied so far. These parameters sufficiently describe the current state of the system before applying the next test. Note that there is only a polynomial number of combinations of these parameters that need to be considered. Then the algorithm has up to $m$ possible options for the next test of box $i$ – or $m$ possible options for the first test of box $i+1$, thereby concluding the testing of box $i$. Hence, there are only polynomially many combinations that need to be considered to find the optimal decision for the current test (assuming that an optimal testing algorithm for the subsequent number of tests/boxes has already been computed via backwards induction).

We can also transfer the approximation guarantee for the algorithm from Theorem 9. We apply the algorithm in the model with probability tests and interpret them as randomized threshold tests. By applying the arguments of Proposition 6 to the sequential model with multiple tests per box, we see that for every randomized threshold testing algorithm there is a deterministic one that performs at least as good. Overall, this yields the following corollary.

▶ **Corollary 10.** *For finite discrete distributions, an optimal sequential testing algorithm for multiple tests per box can be computed by dynamic programming in polynomial time. It is at least* $(1 - o(1))$*-competitive for threshold testing with multiple tests per box.*

## 6 Conclusion

In this paper, we have initiated the study of threshold testing of i.i.d. random variables, a probing model with partial revelation and binary feedback. For non-adaptive algorithms, the model is essentially equivalent to the standard gambler's problem, and optimal performance is governed by the i.i.d. prophet inequality of approximately 0.745. For adaptive algorithms, we obtain a testing algorithm with competitive ratio of 0.869. This significantly outperforms 0.745, proves that there is a substantial adaptivity gap, and reveals the structural difference of the adaptive problem. Moreover, we show a constant upper bound on the ratio achievable by any adaptive testing algorithm. In contrast, when we can (adaptively) apply multiple tests to a single box, it is possible to achieve even a ratio of $1 - o(1)$.

There are many intriguing open problems arising from our work. Obviously, the current upper and lower bounds for the i.i.d. model are not tight. More generally, a simple argument similar to Observation 1 shows that free-order prophet inequalities [5] transfer directly to non-adaptive threshold testing, even for non-i.i.d. boxes. It is an intriguing open problem whether these guarantees can be strictly improved using an adaptive testing algorithm. Can we obtain a ratio strictly larger than 0.745 also for non-i.i.d. threshold testing?

In addition, there are many combinatorial versions of the problem that deserve attention, i.e., when the algorithm is allowed to select more than one box. Testing algorithms for, e.g., knapsack, matroid, or general downward-closed feasibility structures represent a natural and important direction for future research.

## References


1 Marek Adamczyk, Maxim Sviridenko, and Justin Ward. Submodular stochastic probing on matroids. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 29–40, 2014.

2 S. Agrawal, J. Sethuraman, and X. Zhang. On optimal ordering in the optimal stopping problem. In *ACM Conference on Economics and Computation (EC)*, pages 187–188, 2020.

3 Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. *SIAM J. Comput.*, 43(2):930–972, 2014.

4 Arash Asadpour, Hamid Nazerzadeh, and Amin Saberi. Stochastic submodular maximization. In *Workshop on Internet and Network Economics (WINE)*, pages 477–489, 2008.

5 Archit Bubna and Ashish Chiplunkar. Prophet inequality: Order selection beats random order. *CoRR*, abs/2211.04145, 2022. `arXiv:2211.04145`.

6 José R. Correa, Andrés Cristi, Boris Epstein, and José A. Soto. Sample-driven optimal stopping: From the secretary problem to the i.i.d. prophet inequality. *CoRR*, abs/2011.06516, 2020. `arXiv:2011.06516`.

7 José R. Correa, Paul Dütting, Felix A. Fischer, and Kevin Schewior. Prophet inequalities for independent and identically distributed random variables from an unknown distribution. *Math. Oper. Res.*, 47(2):1287–1309, 2022.

8 José R. Correa, Paul Dütting, Felix A. Fischer, Kevin Schewior, and Bruno Ziliotto. Unknown I.I.D. prophets: Better bounds, streaming algorithms, and a new impossibility (extended abstract). In *Innovations in Theoretical Computer Science Conference (ITCS)*, pages 86:1–86:1, 2021.

9 José R. Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. Recent developments in prophet inequalities. *SIGecom Exch.*, 17(1):61–70, 2018.

10 José R. Correa, Patricio Foncea, Ruben Hoeksma, Tim Oosterwijk, and Tjark Vredeveld. Posted price mechanisms and optimal threshold strategies for random arrivals. *Math. Oper. Res.*, 46(4):1452–1478, 2021.

11 José R. Correa, Raimundo Saona, and Bruno Ziliotto. Prophet secretary through blind strategies. *Math. Program.*, 190(1):483–521, 2021.

12 Paul Dütting, Michal Feldman, Thomas Kesselheim, and Brendan Lucier. Prophet inequalities made easy: Stochastic optimization by pricing nonstochastic inputs. *SIAM J. Comput.*, 49(3):540–582, 2020.

13 Oliver Göbel, Martin Hoefer, Thomas Kesselheim, Thomas Schleiden, and Berthold Vöcking. Online independent set beyond the worst-case: Secretaries, prophets, and periods. In *International Colloqium on Automata, Languages, and Programming (ICALP)*, pages 508–519, 2014.

14 Anupam Gupta, Haotian Jiang, Ziv Scully, and Sahil Singla. The markovian price of information. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 233–246, 2019.

15 Anupam Gupta and Viswanath Nagarajan. A stochastic probing problem with applications. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 205–216, 2013.

16 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Algorithms and adaptivity gaps for stochastic probing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1731–1747, 2016.

17 Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. Adaptivity gaps for stochastic probing: Submodular and XOS functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1688–1702. SIAM, 2017.

18 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 58–65, 2007.

19 T. P. Hill and R. P. Kertz. Comparisons of stop rule and supremum expectations of i.i.d. random variables. *Annals of Probability*, 10(2):336–345, 1982.

**20** Martin Hoefer, Kevin Schewior, and Daniel Schmand. Stochastic probing with increasing precision. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4069–4075, 2021.

**21** Robert Kleinberg and S. Matthew Weinberg. Matroid prophet inequalities and applications to multi-dimensional mechanism design. *Games Econ. Behav.*, 113:97–115, 2019.

**22** U. Krengel and L. Sucheston. Semiamarts and finite values. *Bull. Amer. Math. Soc.*, 83:745–747, 1977.

**23** Renato Paes Leme, Balasubramanian Sivan, Yifeng Teng, and Pratik Worah. Pricing query complexity of revenue maximization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 399–415. SIAM, 2023.

**24** Bo Li, Xiaowei Wu, and Yutong Wu. Query efficient prophet inequality with unknown I.I.D. distributions. *CoRR*, abs/2205.05519, 2022. `arXiv:2205.05519`.

**25** Allen Liu, Renato Paes Leme, Martin Pál, Jon Schneider, and Balasubramanian Sivan. Variable decomposition for prophet inequalities and optimal ordering. In *ACM Conference on Economics and Computation (EC)*, page 692, 2021.

**26** Brendan Lucier. An economic view of prophet inequalities. *SIGecom Exch.*, 16(1):24–47, 2017.

**27** Michela Meister and Sloan Nietert. Learning with comparison feedback: Online estimation of sample statistics. In *Algorithmic Learning Theory (ALT)*, volume 132, pages 983–1001, 2021.

**28** Princewill Okoroafor, Vaishnavi Gupta, and Robert Kleinberg. Non-stochastic CDF estimation using threshold queries. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3551–3572. SIAM, 2023.

**29** Bo Peng and Zhihao Gavin Tang. Order selection prophet inequality: From threshold optimization to arrival time design. In *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 171–178, 2022.

**30** Sebastian Perez-Salazar, Mohit Singh, and Alejandro Toriello. The IID prophet inequality with limited flexibility. *CoRR*, abs/2210.05634, 2022. `arXiv:2210.05634`.

**31** Aviad Rubinstein, Jack Z. Wang, and S. Matthew Weinberg. Optimal single-choice prophet inequalities from samples. In *Innovations in Theoretical Computer Science Conference (ITCS)*, pages 60:1–60:10, 2020.

**32** Martin L. Weitzman. Optimal search for the best alternative. *Econometrica*, 47:641–654, 1979.

# Parameterized Complexity of Fair Bisection

## FPT-Approximation meets Unbreakability

**Tanmay Inamdar**
University of Bergen, Norway

**Daniel Lokshtanov**
University of California Santa Barbara, CA, USA

**Saket Saurabh**
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Norway

**Vaishali Surianarayanan**
University of California Santa Barbara, CA, USA

## Abstract

In the Minimum Bisection problem input is a graph $G$ and the goal is to partition the vertex set into two parts $A$ and $B$, such that $||A| - |B|| \leq 1$ and the number $k$ of edges between $A$ and $B$ is minimized. The problem is known to be NP-hard, and assuming the Unique Games Conjecture even NP-hard to approximate within a constant factor [Khot and Vishnoi, J.ACM'15]. On the other hand, a $\mathcal{O}(\log n)$-approximation algorithm [Räcke, STOC'08] and a parameterized algorithm [Cygan et al., ACM Transactions on Algorithms'20] running in time $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ is known.

The Minimum Bisection problem can be viewed as a clustering problem where edges represent similarity and the task is to partition the vertices into two equally sized clusters while minimizing the number of pairs of similar objects that end up in different clusters. Motivated by a number of egregious examples of unfair bias in AI systems, many fundamental clustering problems have been revisited and re-formulated to incorporate fairness constraints. In this paper we initiate the study of the Minimum Bisection problem with fairness constraints. Here the input is a graph $G$, positive integers $c$ and $k$, a function $\chi : V(G) \to \{1, \ldots, c\}$ that assigns a color $\chi(v)$ to each vertex $v$ in $G$, and $c$ integers $r_1, r_2, \cdots, r_c$. The goal is to partition the vertex set of $G$ into two almost-equal sized parts $A$ and $B$ with at most $k$ edges between them, such that for each color $i \in \{1, \ldots, c\}$, $A$ has exactly $r_i$ vertices of color $i$. Each color class corresponds to a group which we require the partition $(A, B)$ to treat fairly, and the constraints that $A$ has exactly $r_i$ vertices of color $i$ can be used to encode that no group is over- or under-represented in either of the two clusters.

We first show that introducing fairness constraints appears to make the Minimum Bisection problem qualitatively harder. Specifically we show that unless FPT=W[1] the problem admits no $f(c)n^{\mathcal{O}(1)}$ time algorithm even when $k = 0$. On the other hand, our main technical contribution shows that is that this hardness result is simply a consequence of the very strict requirement that each color class $i$ has *exactly* $r_i$ vertices in $A$. In particular we give an $f(k, c, \epsilon)n^{\mathcal{O}(1)}$ time algorithm that finds a balanced partition $(A, B)$ with at most $k$ edges between them, such that for each color $i \in [c]$, there are at most $(1 \pm \epsilon)r_i$ vertices of color $i$ in $A$.

Our approximation algorithm is best viewed as a proof of concept that the technique introduced by [Lampis, ICALP'18] for obtaining FPT-approximation algorithms for problems of bounded tree-width or clique-width can be efficiently exploited even on graphs of unbounded width. The key insight is that the technique of Lampis is applicable on tree decompositions with unbreakable bags (as introduced in [Cygan et al., SIAM Journal on Computing'14]). An important ingredient of our approximation scheme is a combinatorial result that may be of independent interest, namely that for every $k$, every graph $G$ admits a tree decomposition with adhesions of size at most $\mathcal{O}(k)$, unbreakable bags, and logarithmic depth.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms



31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 63;
pp. 63:1–63:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Keywords and phrases** FPT Approximation, Minimum Bisection, Unbreakable Tree Decomposition, Treewidth

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.63

**Related Version** *Full Version*: https://arxiv.org/abs/2308.10657

**Funding** *Tanmay Inamdar*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
*Daniel Lokshtanov*: NSF award CCF-2008838.

*Saket Saurabh*: European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA01/2017-18.
*Vaishali Surianarayanan*: NSF award CCF-2008838.

## 1 Introduction

Clustering is one of the most fundamental problems in computer science. In a clustering problem, we are typically interested in dividing the given collection of data points into a group of *clusters*, such that the set of data points belonging to each cluster are more "similar" to each other, as compared to the points belonging to other clusters. Depending on the specific setting and application, there are a number of ways to model this abstract task of clustering as a concrete mathematical problem. We refer the reader to surveys such as [30, 27, 3] for a detailed background and literature on the topic.

In one such model of clustering, the input is represented as a simple, undirected graph, and the existence of an edge between a pair of vertices denotes that the two vertices are related to, or similar to, each other. For example, this is how one models social networks as graphs [25] – the set of vertices corresponds to people, and an edge represents that the two people are friends with each other. In this setting, the classical Minimum Bisection problem can be thought of as a clustering problem [31, 7] – we are interested in finding two size-balanced clusters of vertices, such that the number of edges going across the two clusters is minimized. More formally, in Minimum Bisection problem, we are given a graph $G = (V, E)$ on $n$ vertices, and a non-negative integer $k$, and the goal is to determine whether there exists a balanced edge cut $(A, B)$ of order $k$. Here, an *edge cut* $(A, B)$ is a partition of $V(G)$ into two non-empty subsets $A$ and $B$, an edge cut is *balanced* if $||A| - |B|| \leq 1$, and the *order* of the cut $(A, B)$ is the number of edges with one endpoint in $A$ and the other in $B$. The NP-completeness of Minimum Bisection has long been known [13], and it is extensively studied from the perspective of approximation and parameterized algorithms. Minimum Bisection admits a logarithmic approximation in polynomial time [26], and it is hard to approximate within any constant factor, assuming the Unique Games Conjecture [19]. In the realm of Parameterized Algorithms, one can solve the problem exactly in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$, i.e., it is Fixed-Parameter Tractable (FPT) parameterized by $k$ [10, 9].

More recently, the notion of *fairness* has gained prominence in the literature of clustering algorithms – and algorithm design in general. This is motivated from the fact that, often the real-life data reflects unconscious biases, and unless the algorithm is explicitly required to counteract these biases, the output of the algorithm may have real-life consequences that are *unfair* ((see, e.g., [15, 24, 11]). Researchers have proposed different models of fairness for the traditional center-based clustering problems, such as $k$-Median/Means/Center. These models of fairness can be broadly classified into two types – *individual fairness*, and *group fairness*. At a high level, individual fairness requires that the solution treats each of

the individuals (a point) in a fair way, e.g., every point has a cluster-center "nearby" [6]. On the other hand, in the group fairness setting, the set of points is typically divided into multiple colors, where each color represents, say a particular demographic (such as gender, ethnicity etc.). In this setting, the fairness constraints are represented in terms of the colors as a group. There are multiple notions of group fairness (see, e.g., [2, 6, 12, 22, 14, 18]), but to the specific interest to us is the *color-balanced clustering* model, studied in [28, 17, 1]. Roughly speaking, in this setting we want the "local proportions" of all colors in every cluster to be approximately equal to their "global proportions". Inspired from this *color-balanced* notion of fairness, study the following *fair* version of Minimum Bisection.

In this problem formulation the color classes $i \in \{1, \ldots, c\}$ are protected groups which are required to be treated fairly by the clustering algorithm. The imposed fairness constraint for group $i$ is that, in the edge cut $(A, B)$, the set $A$ contains precisely $r_i$ vertices colored $i$.

Fair Bisection
**Input:** An instance $(G, c, k, r^\circ, \chi)$, where
- $G$ is an unweighted graph
- $c$ and $k$ are positive integers
- $\chi : V(G) \to c$ is a coloring function on $V(G)$ using at most $c$ colors
- $r^\circ = (r_1, \cdots, r_c)$ is a $c$ length tuple of positive integers

**Question:** Does there exist an edge cut $(A, B)$ of $G$ of order at most $k$ having exactly $r_i$ vertices of color $i$ in $A$ for each $i \in [c]$.

We will say that an edge cut that satisfies the fairness constraints imposed by the tuple $r^\circ$ is $r^\circ$-*fair*.

Thus, when $r_i$ is set to be precisely half of the number $c_i$ of vertices colored $i$ an $r^\circ$-fair edge cut must evenly split each color class across the two sides $A$ and $B$.

## Our Results

It is quite easy to see that the existing parameterized algorithms [9, 10] for Minimum Bisection directly generalize to a $n^{\mathcal{O}(c)}k^{\mathcal{O}(k)}$ time algorithm for Fair Bisection[1]. Therefore, the first natural question is whether it is possible to eliminate the dependence on $c$ in the exponent of $n$ in the running time. Our first result (Theorem 20) is that, assuming $\mathsf{FPT} \neq \mathsf{W}[1]$, an $f(c)n^{\mathcal{O}(1)}$ time algorithm is not possible even when $k = 0$. In fact, this hardness result holds even in the special case where the vertices of each color are required to be evenly split across both partitions (in particular, when $2r_i = c_i$ for every $i$).

Our main technical contribution (Theorem 11) is to show that this hardness result is quite brittle. Indeed, the requirement that each color class $i$ have *exactly* $r_i$ vertices in $A$ is probably much too strong in the color-balanced fairness setting. We are satisfied even if the number of vertices of each color class is sufficiently close to the desired target number. We will say that an edge cut $(A, B)$ is $(\epsilon, r^\circ)$-*fair* if $A$ contains no more than $r_i(1 + \epsilon)$ of vertices colored $i$ and $B$ contains no more than $(c_i - r_i)(1 + \epsilon)$ vertices colored $i$. We show (in Theorem 11) that there exists an algorithm that takes as input an instance $(G, c, k, r^\circ, \chi)$, together with an $\epsilon > 0$, runs in time $f(\epsilon, k, c)n^{\mathcal{O}(1)}$, and if $G$ has a $r^\circ$-fair edge cut $(\hat{A}, \hat{B})$ of order at most $k$ then the algorithm produces a $(\epsilon, r^\circ)$-fair edge cut $(A, B)$ of order at most $k$.

[1] A formal proof of this claim is a corollary of our Theorem 11.

## Our Methods

The hardness result of Theorem 20 is a fairly straightforward parameterized reduction from Multi-Dimensional Subset Sum parameterized by the dimension[2], whose main purpose is to put the parameterized approximation scheme of Theorem 11 in context. We only discuss here the methods in the proof of of Theorem 11.

At a *very* high level the algorithm of Theorem 11 is the combination of two well-known techniques in parameterized algorithms: dynamic programming over tree decompositions with unbreakable bags (introduced by Cygan et al. [10]), and the geometric rounding technique of Lampis [20] for parameterized approximation schemes for problems on graphs of bounded tree-width or clique-width. The conceptual novelty in (and perhaps the most interesting technical aspect of) our work is to realize that Lampis' technique can be applied even to dynamic programming algorithms over tree decompositions with unbounded width to yield approximation schemes for parameterized problems on general graphs. Executing on this vision requires a few non-trivial technical insights, which we will shortly highlight. However, to describe these technical insights in more detail we first give a brief description of the two techniques that we combine.

### Lampis' Geometric Rounding Technique

We first discuss how the technique of Lampis [20] applies to tree decompositions of bounded width. A *tree decomposition* of a graph $G$ is a pair $(T, \beta)$ where $T$ is a tree and $\beta$ is a function that assigns to each vertex $t \in V(T)$ a vertex set $\beta(t) \subseteq V(G)$ (called a *bag*) in $G$. To be a tree decomposition the pair $(T, \beta)$ must satisfy the tree-decomposition axioms: *(i)* for every $v \in V(G)$ the set $\{t \in V(T) \ : \ v \in \beta(t)\}$ induces a non-empty and connected subgraph of $T$, and *(ii)* for every edge $uv \in E(G)$ there exists a $t \in V(T)$ such that $\{u, v\} \subseteq \beta(t)$. The *width* (or tree-width) of a decomposition $(T, \beta)$ is defined as $\max_{t \in V(T)} |\beta(t)| - 1$.

Roughly speaking, Lampis' technique considers dynamic programming (DP) algorithms over a tree decomposition $(T, \beta)$ of $G$ of width $k$. In such an algorithm there is a DP-table for every node $t$ of the decomposition tree, and suppose that the entries in these tables are indexed by vectors in $\{1, 2, \ldots, n\}^d$ (for some integer $d$), where $n$ is the number of vertices of $G$. To decrease the size of the DP tables and thereby also the running time of the algorithm, one "sparsifies" the DP table to only consider entries in $S^d$, where $S = \left\{\lfloor (1+\delta)^i \rfloor : i \geq 0\right\}$. This makes the size of the DP table upper bounded by $(\log_{1+\delta} n)^{\mathcal{O}(d)}$, at the cost of introducing a multiplicative error of $(1+\delta)$ in every round of the DP algorithm (since now vectors in $\{1, 2, \ldots, n\}^d$ are "approximated" by their closest vector in $S^d$). If the decomposition tree $T$ has depth $\mathcal{O}(\log n)$ the dynamic program only needs $\mathcal{O}(\log n)$ rounds, and so the total error of the algorithm is a multiplicative factor of $(1+\delta)^{\mathcal{O}(\log n)}$. Setting $\delta = \epsilon / \log^2 n$ gives the desired trade-off between DP table size (and therefore running time) and accuracy. Luckily, every tree decomposition of width $k$ can be turned into a tree decomposition of width at most $3k+2$ and depth $\mathcal{O}(\log n)$ [4] and so this approach works on all graphs of tree-width $k$.

### Tree Decompositions with Unbreakable Bags

We now turn to the technique of Cygan et al. [10] for Minimum Bisection, namely dynamic programming over tree decompositions with small adhesions and unbreakable bags. We again need to define a few technical terms. An *adhesion* of a tree decomposition $(T, \beta)$ of a graph

[2] The hardness of Multi-Dimensional Subset Sum parameterized by the dimension is folklore, but we were unable to find a reference, so for completeness we provide a proof.

$G$ is a set $\beta(u) \cap \beta(v)$ for an edge $uv \in E(T)$. The *adhesion size* of a tree-decomposition is just the maximum size of an adhesion of the decomposition. A tree decomposition $(T, \beta)$ is said to have $(q, k)$-*unbreakable bags* if for every bag $\beta(t)$ of the decomposition and every edge-cut $(A, B)$ of order at most $k$ in $G$ it holds that $\min(|A \cap \beta(t)|, |B \cap \beta(t)|) \leq q$.

The main engine behind the algorithm of Cygan et al. [10] (see also [9]) is a structural theorem that for every graph $G$ and integer $k$ there exists a tree decomposition $(T, \beta)$ of $G$ with adhesion size at most $k$ and $(k+1, k)$-unbreakable bags. This is coupled with an observation that even though this tree decomposition might have unbounded tree-width, we can still do dynamic programming over this tree decomposition, keeping a DP table for every *adhesion* of the tree decomposition, rather than for every bag. However, while tree-width based DP algorithms utilize a simple recurrence to calculate the DP table at a bag from the tables of its children, Cygan et al. [10] need to turn to a clever "randomized contraction" (see [8]) based algorithm to compute the DP table for an adhesion from the DP tables of its children.

### Combining Tree Decompositions with Unbreakable Bags and Geometric Rounding

As we mentioned eariler, the technique of Cygan et al. [10] for Minimum Bisection generalizes in a relatively straightforward way, to give a $f(k)n^{\mathcal{O}(c)}$ time algorithm for Fair Bisection. Here we do dynamic programming over the tree decomposition of $G$ with adhesions of size $k$ and $(k+1, k)$-unbreakable bags. We have a DP table for every adhesion that is indexed by a vector in $[n]^c$ (this vector describes partial solutions, where the $i^{th}$ element of the vector is the number of vertices of color $i$ that have so far been put on the $A$ side in this partial solution).

We want to apply Lampis' geometric rounding technique and "sparsify" the DP table to only consider entries in $S^c$, where $S = \left\{\lfloor (1+\delta)^i \rfloor : i \geq 0\right\}$. There are a few technical obstacles to realizing this plan, that we overcome. The most important one of them is that the depth reduction theorem of Bodlaender and Hagerup [4] only applies to tree decompositions of bounded width, therefore it is not immediate how to obtain a tree decomposition with small adhesions, unbreakable bags and logarithmic depth. A closer inspection of the proof sketch of Bodlaender and Hagerup [4] reveals that a tree decomposition with adhesions of size $k$ and $(k+1, k)$-unbreakable bags can be turned into a tree decomposition with adhesions of size $\mathcal{O}(k)$, and logarithmic depth, such that each bag of the new decomposition is the union of a constant number of bags of the old one (the bags in this new decomposition do *not* need to themselves be unbreakable). Nevertheless we prove that some careful modifications to this tree decomposition are sufficient to obtain a tree decomposition with adhesions of size $\mathcal{O}(k)$, logarithmic depth, and $(\mathcal{O}(k), k)$-unbreakable bags (see Theorem 9). We believe that Theorem 9 will be a useful tool for future applications of Lampis' geometric rounding technique to tree decompositions with unbreakable bags.

### Organization of the Paper

We begin by defining the basic notions on graphs and tree decompositions in Section 2. In Section 3, we prove Corollary 10 that shows how to obtain logarithmic-depth unbreakable tree decompositions. Then, in Section 4, we use such a tree decomposition to design our exact and approximate algorithms. In Section 5, we sketch the proof of our hardness result, which shows that Fair Bisection is W[1]-hard parameterized by $c$ even when $k = 0$. Finally, in Section 6, we give concluding remarks and future directions. Proofs marked with $*$ can be found in the full version of the paper.

## 2 Preliminaries

For an integer $k$, we denote the set $\{1, 2, \ldots, k\}$ by $[k]$. For a graph $G$, *an edge cut* is a pair $A, B \subseteq V(G)$ such that $A \cup B = V(G)$ and $A \cap B = \emptyset$. The order of an edge cut $(A, B)$ is $|E(A, B)|$, that is, the number of edges with one endpoint in $A$ and the other in $B$. For a subset $X \subseteq V(G)$, let $G \setminus X$ denote the graph $G[V(G) \setminus X]$. For an edge cut $(A, B)$, and a subset $X \subseteq V(G)$, the cut *induced* on $X$ by $(A, B)$ is $(A \cap X, B \cap X)$.

▶ **Definition 1** (unbreakability). *A set $X \subseteq V(G)$ is* $(q, s)$-edge-unbreakable *if every edge cut $(A, B)$ of order at most $s$ satisfies $|A \cap X| \leq q$ or $|B \cap X| \leq q$.*

For a rooted tree $T$ and vertex $t \in V(T)$, we denote by $T_t$ the subtree of $T$ rooted at $t$. For a rooted tree $T$ and a non-root vertex $t \in V(T)$, we denote the parent of $t$ by $\mathcal{P}(t)$. The depth of a tree $T_t$ is the maximum length of a $t$ to leaf path in $T_t$. For a node $t$, we denote $\mathsf{ht}_T(t)$ to be the the depth of the subtree $T_t$ rooted at $t$ in $T$.

Consider a tree decomposition $(T, \beta)$ of a graph $G$. For every $t \in V(T)$ a set $\beta(t) \subseteq V(G)$, is called a *bag*. We can extend the function $\beta$ to subsets of $V(T)$ in the natural way: for a subset $X \subseteq V(T)$, $\beta(X) := \bigcup_{x \in X} \beta(x)$. Another important notion that we need is of tree decomposition where bags are "highly connected", i.e., unbreakable. For a rooted tree $T$ and vertex $v \in V(T)$ we denote by $T_v$ the subtree of $T$ rooted at $v$. We refer to the vertices of $T$ as nodes.

For $s, t \in V(T)$ we say that *$s$ is a descendant of $t$* or that *$t$ is an ancestor of $s$* if $t$ lies on the unique path from $s$ to the root; note that a node is both an ancestor and a descendant of itself. By $\mathsf{child}(t)$, we denote the set of children of $t$ in $T$. For any $X \subseteq V(T)$, define $G_X := G[\cup_{t \in X} \beta(V(T_t))]$.

We define an *adhesion* of an edge $e = (t, t_0) \in E(T)$ to be the set $\sigma(e) := \beta(t) \cap \beta(t_0)$, and an adhesion of $t \in V(T)$ to be $\sigma(t) := \sigma(t, \mathcal{P}(t))$, or $\sigma(t) = \emptyset$ if the parent of $t$ does not exist, i.e., when $t$ is the root of $T$. We define the following notation for convenience:

$$\gamma(t) := \bigcup_{s\text{: descendant of } t} \beta(s)$$

$$\alpha(t) := \gamma(t) \setminus \sigma(t), \qquad G_t := G[\gamma(t)] - E(G[\sigma(t)]).$$

We say that a rooted tree decomposition $(T, \beta)$ of $G$ is *compact* if for every node $t \in V(T)$ for which $\alpha(t) \neq \emptyset$ we have that $G[\alpha(t)]$ is connected and $N_G(\alpha(t)) = \sigma(t)$.

## 3 Obtaining a Low Depth Unbreakable Tree Decomposition

In this section we show that there exists a tree decomposition that has low (i.e., $\mathcal{O}(\log n)$) depth, small-size (i.e., $\mathcal{O}(k)$) adhesions, and $(\mathcal{O}(k), k)$-unbreakable bags. To this end, we design a polynomial-time algorithm that, given a tree decomposition with small adhesions and unbreakable bags, produces a tree decomposition with the aforementioned properties. In the next section, we design a dynamic programming algorithm over such a low depth decomposition to obtain an FPT approximation for Fair Bisection.

In our algorithm, we use the notion of a *tree partition* of a graph, which, informally, captures the "tree-likeness" of a graph. Tree partitions were introduced by [29, 16], and are easy to define.

▶ **Definition 2** (Tree Partition). *A tree partition of a graph $G$ is a pair $(\mathcal{T}, \tau)$ where $\mathcal{T}$ is a tree and $\tau : V(G) \to V(\mathcal{T})$ is a function from $V(G)$ to $V(\mathcal{T})$ such that for each $e = (u, v) \in E(G)$ either $\tau(u) = \tau(v)$ or $(\tau(u), \tau(v)) \in E(T)$. A rooted tree partition $(\mathcal{T}, \tau)$ with root $r$ is the tree partition $(\mathcal{T}, \tau)$ where the tree $\mathcal{T}$ is a rooted tree with root $r$.*

We remark that we use calligraphic font ($\mathcal{T}$) to denote trees corresponding to Tree Partitions to easily distinguish them from graphs that are trees. Observe that for a tree $T$, the pair $(\mathcal{T} = T, \tau)$ where $\tau(v) = v$ for each $v \in T$ is a trivial tree partition of $T$. For our result we only use tree partitions of trees. Given a tree decomposition $(T, \beta)$ with small adhesions and unbreakable bags, our goal in this section is to use $(T, \beta)$ to obtain a tree decomposition of bounded height without blowing up the adhesion size and unbreakability guarantees too much. For this we first find a tree partition $(\mathcal{T}, \tau)$ of $T$ that satisfies additional properties, such as logarithmic depth and for each $t \in V(\mathcal{T})$, it holds that $|\tau^{-1}(t)| \leq 4$. Using this tree partition, we obtain a tree decomposition $(\mathcal{T}, \beta_1)$ whose underlying tree is $\mathcal{T}$, and each bag $\beta_1(t)$, $t \in V(\mathcal{T})$ is a union of at most 4 bags of $(T, \beta)$; $\beta_1(t) = \bigcup_{x \in \tau^{-1}(t)} \beta(x)$. This tree decomposition already has bounded height, small adhesion size and each bag is a union of at most four unbreakable bags of $(T, \beta)$. From here with some extra work we obtain a tree decomposition with unbreakable bags as well. For this we use other properties of $(\mathcal{T}, \tau)$ to modify $(\mathcal{T}, \beta_1)$ to obtain our desired tree decomposition. As outlined above, for our result we need tree partitions of a tree satisfying some properties. We now define such tree partitions below and show how to find one in polynomial time.

▶ **Definition 3** (Nice Tree Partition). *A tree partition $(\mathcal{T}, \tau)$ of a tree $T$ is said to be a nice tree partition if it satisfies the following properties:*

1. *$\mathcal{T}$ has depth at most $\lceil \log_2 |V(T)| \rceil$*
2. *for each $t \in V(\mathcal{T})$, $1 < |\tau^{-1}(t)| \leq 4$.*
3. *for each $t \in V(\mathcal{T})$, $T[V_t]$ is a subtree of $T$, where $V_t = \bigcup_{x \in V(\mathcal{T}_t)} \tau^{-1}(x)$.*

We now show how to find a nice tree partition of a tree in polynomial time. For this we use a recursive procedure. The core idea in each recursive step is to map a balanced separator $b$ of the tree to the root of the tree partition. To ensure the connectivity properties of a tree partition, we have a set $M$ of marked vertices in the tree that are always mapped to the root of the tree partition in addition to $b$. Then for each connected component in the forest obtained by removing $M \cup \{b\}$ from the tree, we mark new vertices and recurse. We need some extra work to make sure that every node in the tree partition is mapped to by only a constant number of nodes in the tree. For this we ensure that in each recursive call we mark only a few ($\leq 2$) new vertices.

▶ **Lemma 4.** *Given a tree $T$ on $n$ vertices with root $r$, one can in polynomial time compute a rooted nice tree partition $(\mathcal{T}, \tau)$ of $T$ with root $r_{\mathcal{T}}$ such that $\tau(r) = r_{\mathcal{T}}$.*

**Proof.** We now design a procedure `FindBalancedTP` that takes as an argument a tree $T'$, and a non-empty set $M \subseteq V(T')$ of size at most 2, and returns a rooted nice tree partition $(\mathcal{T}', \tau')$ of $T'$ with root $r'$, such that $M \subseteq \tau'^{-1}(r')$. We will invoke this procedure on the input tree $T$ with $M = \{r\}$, where $r$ is the root of $T$ to obtain a rooted nice tree partition $(\mathcal{T}, \tau)$ of $T$.

In the procedure `FindBalancedTP`$(T', M)$ we carry out the following steps:

- We find a balanced bisector $b$ of $T'$ and initialize $M' = M \cup \{b\}$.
- If all vertices in $M'$ do not lie on a path in $T'$, we add an extra vertex $x$ to $M'$. Let $x$ be the last common vertex on the path from $m_1$ to $m_2$ and the path from $m_1$ to $b$ in $T$. Modify $M' = M' \cup \{x\}$.
- For each tree $H$ in the forest $T' \setminus M'$, we recursively call `FindBalancedTP`$(H, M_H)$ where $M_H = N_{T'}(M') \cap V(H)$ is the set of neighbors of vertices in $M'$ in $H$. Let $(\mathcal{H}, \tau_H)$ be the tree partition returned by this procedure call.

- We now construct a tree partition $(\mathcal{T}', \tau')$ with root $r'$. We assign $\tau'^{-1}(r') = M'$. Then for each tree $H$ in the forest $T' \setminus M'$, we make $\mathcal{H}$ a subtree of $\mathcal{T}'$ by attaching the root of $\mathcal{H}$ as a child to $r'$. Further for each $t \in V(\mathcal{H})$ we assign $\tau'^{-1}(t) = \tau_H(t)$.
- We return $(\mathcal{T}', \tau')$.

We now prove that for any tree $T'$ with root $r'$, and any non-empty subset $M \subseteq V(T')$ with $0 < |M| \leq 2$, the procedure `FindBalancedTP`$(T', M)$ returns a rooted nice tree partition $(\mathcal{T}', \tau')$ of $T'$ with root $r'$ such that $M \subseteq \tau'^{-1}(r')$. The proof is by induction.

*Base Case $|V(T')| = 1$ or $V(T') = M'$:* In this case $V(\mathcal{T}') = \{r'\}$ and $\tau(r') = M'$. Observe that $0 < |M'| \leq 4$. This is because the procedure is called with a non-empty set $M$ of size at most two. Then, the procedure initializes $M' = M$, and adds at most two other vertices ($b$ and $x$) in $V(T')$ to $M'$. Thus $(\mathcal{T}', \tau')$ is a nice tree partition with root $r'$ and $M \subseteq \tau'^{-1}(r')$.

Now we prove the *inductive case* where $V(T')$ has size $i$, $i > 1$ and $V(T') \neq M'$. For this we assume the inductive hypothesis that the procedure returns a tree partition with the desired properties for all trees $H$ having less than $i$ vertices and non-empty sets $M' \subseteq V(H)$ of size at most two.

Let $H$ be a tree in the forest $T' \setminus M'$. We now show that $|V(H)| \leq \lceil |V(T')|/2 \rceil$ and $1 < |M_H| \leq 2$. By construction $M'$ contains the vertex $b$, a balanced bisector of $T'$. Thus $V(H)$ has size at most $\lceil |V(T')|/2 \rceil$. $|M_H| > 1$ since $H$ contains at least one child of $M'$ since it is a tree in the forest $T' \setminus M'$. To show $|M_H| \leq 2$, we first show there is a vertex $s$ in $M'$ such that in the forest $T' \setminus \{s\}$ every vertex $s' \in M' \setminus \{s\}$ is contained in a different tree. If all vertices in $M \cup \{b\}$ do not lie on a path in $T'$, then $s$ is just the vertex $x$ we added to $M'$ in the second step of the procedure. If the vertices of $M \cup \{b\}$ lie on a path $P$ in $T'$ then $M' = M \cup \{b\}$. In this case if $|M'| \leq 2$, then $s$ is any vertex in $M'$. On the other hand if $|M'| = 3$, then $s$ is the second vertex from $M'$ in the path $P$. Due to the property of $s$, $H$ may contain a child of $s$ and a child of one other $s' \in M' \setminus \{s\}$. Thus $|M_H| \leq 2$.

Since $H$ is a tree with $|V(H)| \leq \lceil |V(T')|/2 \rceil$ and $1 < |M_H| \leq 2$, by induction the tree partition $(\mathcal{H}, \tau_H)$ returned by the call to the procedure `FindBalancedTP`$(H, M_H)$ is a nice tree partition with root $r_H$ and $M_H \subseteq \tau_H^{-1}(r_H)$.

We now show that $(\mathcal{T}, \tau')$ is a rooted tree partition of $T'$ with root $r'$. First we show that each vertex $v \in \mathcal{T}$ is mapped to exactly one vertex $t \in \mathcal{T}$ by $\tau'$. If $v \in M'$, then $\tau(v)$ is mapped to $r'$. If $v \in H$, $H \in T' \setminus M'$, then since $(\mathcal{H}, \tau_H)$ is a rooted tree partition of $H$, $\tau(v) = \tau_H(v)$ by construction. Next we show that each edge $(x, y) \in T'$ satisfies either $\tau'(x) = \tau'(y)$ or $(\tau'(x), \tau'(y)) \in E(\mathcal{T})$, by considering three cases. (i) If $x, y \in M'$, then this is trivially true. (ii) If $x, y \notin M'$ then $x, y$ must belong to some tree $H \in T' \setminus M'$ and thus by induction $(\tau'(x) = \tau_H(x), \tau'(y) = \tau_H(y)) \in E(\mathcal{T})$. (iii) If $x \in M'$ and $y \notin M'$, by construction, $\tau(x) = r'$ and $y \in M_H$ for some $H \in T' \setminus M'$. Since $M_H \subseteq \tau_H^{-1}(r_H)$ and $r_H$ is a child of $r'$ in $\mathcal{T}'$, $(\tau'(x) = r, \tau'(y) = r_H) \in E(\mathcal{T})$.

$M \subseteq \tau'^{-1}(r')$ just by construction. We now prove properties $(1) - (4)$ in Definition 3 to show that $(\mathcal{T}', \tau')$ is a nice tree partition. Recall that for each $H \in T' \setminus M'$, $H$ is a subtree of $T'$ with $r_H$ being a child of $r'$ in $\mathcal{T}'$. Then, since $|V(H)| \leq \frac{|V(T)|}{2}$, by inductive hypothesis, $\mathcal{H}$ has depth at most $\lceil \log_2(|V(T')|/2) \rceil = \lceil \log_2(|V(T')|) \rceil - 1$. Therefore, $\mathcal{T}'$ has depth $\lceil \log_2 |V(T')| \rceil$, since the addition of the root $r'$ increases the depth by 1. For each $t' \in V(\mathcal{T}')$, $1 < |\tau'^{-1}(t')| \leq 4$ since $1 < |\tau'^{-1}(r')| \leq 4$ and for each $H \in T' \setminus M'$ and for each $t \in V(\mathcal{H})$, $1 < |\tau_H^{-1}(t)| \leq 4$. For each $t' \in V(\mathcal{T}')$, $T'[V_{t'}]$ is a subtree of $T'$, where $V_{t'} = \bigcup_{x \in V(\mathcal{T}'_{t'})} \tau'^{-1}(x)$ because for each $H \in T' \setminus M'$ and $t \in V(\mathcal{H})$, $H[V_t]$ is a subtree of $H$, where $V_t = \bigcup_{x \in V(\mathcal{H}_t)} \tau_H^{-1}(x)$ and $H$ is subtree of $T'$. This completes the proof. ◀

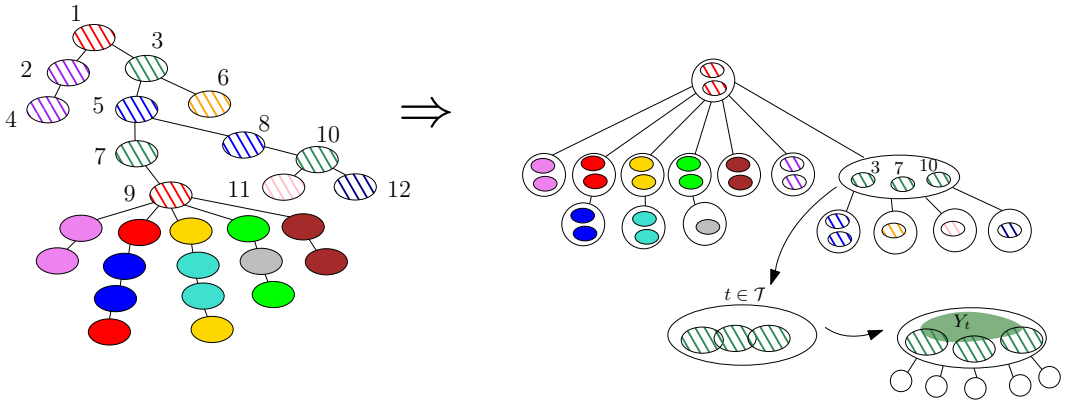



**Figure 1** Left: Tree decomposition $(T, \beta)$ with bags colored for easy understanding. Right: tree decomposition $(\mathcal{T}, \beta_1)$ that is constructed using a tree partition $(\mathcal{T}, \tau)$. The bags in $(T, \beta)$ are mapped according to $\tau$ to $(\mathcal{T}, \beta_1)$. The bags in $\tau^{-1}(t)$, $t \in \mathcal{T}$ can overlap as demonstrated by bags $3, 7, 10$ but their overlap is small and contained in $Y_T$.

Let $(T, \beta)$ be a rooted tree decomposition of a graph $G$ with root $r$, $(q, k)$-unbreakable bags and adhesions of size at most $k$. Further let $(\mathcal{T}, \tau)$ be a rooted nice tree partition of $T$ with root $r_{\mathcal{T}}$ as provided by Lemma 4. We now show that we can obtain a natural rooted tree decomposition $(\mathcal{T}, \beta_1)$ of $G$ where the tree in the decomposition is $\mathcal{T}$. Here $\beta_1 : V(\mathcal{T}) \to 2^{V(G)}$ and $\beta_1(t) = \bigcup_{x \in \tau^{-1}(t)} \beta(x)$. See Figure 1 for an example of $(\mathcal{T}, \beta_1)$.

From now on we fix $G, (T, \beta), (\mathcal{T}, \tau)$, and $\beta_1$ for the rest of the section. We remark that to prove $(\mathcal{T}, \beta_1)$ is a tree decomposition we will not need the *nice* properties of $(\mathcal{T}, \tau)$ nor the properties of the bags and adhesions in $(T, \beta)$. We will later use them to deduce some helpful structural properties of $(\mathcal{T}, \beta_1)$. Figure 1 is the accompanying figure for the proof of the following lemma.

▶ **Lemma 5** $(*)$**.** *The pair $(\mathcal{T}, \beta_1)$ is a tree decomposition of $G$.*

Observe that since $(\mathcal{T}, \tau)$ is a nice tree partition of $T$, the tree decomposition $(\mathcal{T}, \beta_1)$ has depth at most $\lceil \log_2 |V(G)| \rceil$ and each of its bags is a union of at most four bags of $(T, \beta)$. We now prove a few other useful properties of $(\mathcal{T}, \beta_1)$ that will help us design our desired tree decomposition.

▶ **Lemma 6.** *There exists a function $\gamma : V(\mathcal{T}) \to V(T)$, and a set $Y_t \subseteq \beta_1(t)$ for each node $t \in V(\mathcal{T})$, that satisfy, for each node $t \in V(\mathcal{T})$, the following properties:*

1. $|Y_t| \leq 8k$
2. *If $t$ is not the root $r_{\mathcal{T}}$ then $\beta_1(\mathcal{P}(t)) \cap \beta_1(t) \subseteq Y_t$*
3. *for each distinct $x, y \in \tau^{-1}(t)$, $\beta(x) \cap \beta(j) \subseteq Y_t$*
4. *for each child $t_c$ of $t$ in $\mathcal{T}$, it holds that $t = \tau(\gamma(t_c))$ and $\beta_1(t_c) \cap \beta_1(t) \subseteq Y_t \cup \beta(\gamma(t_c))$*

*Furthermore, $\gamma$ and the sets $Y_t$ for each $t \in V(\mathcal{T})$ can be computed in polynomial time.*

**Proof.** For $e = (x, x') \in E(T)$, let $\sigma(e) = \beta(x) \cap \beta(x')$ be the adhesion of edge $e$ in $(T, \beta)$.

For $r_{\mathcal{T}}$, let $T_{r_{\mathcal{T}}} = \bigcup_{x \in \tau^{-1}(r_{\mathcal{T}})} \sigma((x, \mathcal{P}(x)))$ and $\gamma(r_{\mathcal{T}}) = r$. For $t \in V(\mathcal{T})$, $t \neq r_{\mathcal{T}}$, let $E_t = \{e : e = (x, y) \in E(T), x \in \tau^{-1}(\mathcal{P}(t)), y \in \tau^{-1}(t)\}$. Then let $Y_t = \bigcup_{x \in \tau^{-1}(r_{\mathcal{T}})} \sigma((x, \mathcal{P}(x))) \cup \bigcup_{e \in E_T} \sigma(e)$.

For each $x$ in $\tau^{-1}(\mathcal{P}(t))$, there exists at most one $y \in \tau^{-1}(t)$ such that $(x, y) \in E(T)$. This is because $T[V_t]$ is connected, where $t = \bigcup_{x \in V(\mathcal{T}_t)} \tau^{-1}(x)$. If $x$ has an edge to two vertices in $\tau(t)$, then there would be a cycle in $T$. Thus, $|E_t| \leq 4$. Further since $\mathcal{T}$ is from a nice tree partition, $|\tau^{-1}(t)| \leq 4|$. Therefore $|Y_t| \leq 8k$ for each $t \in V(\mathcal{T})$.

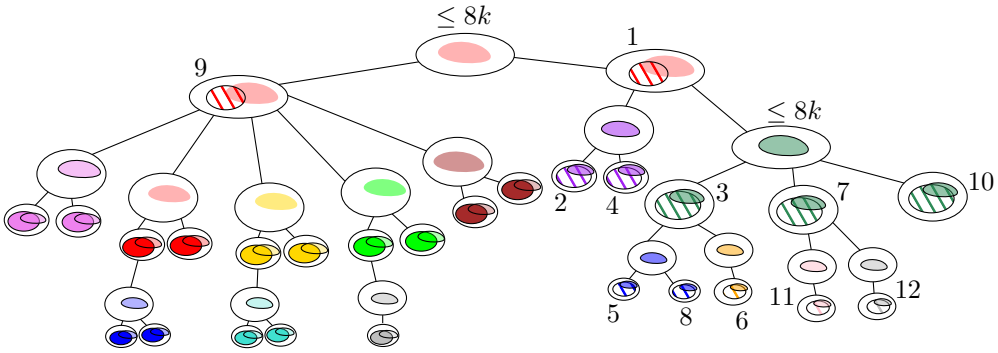



**Figure 2** This shows the final tree decomposition $(T^*, \beta^*)$ constructed using $(T, \beta)$ and $(\mathcal{T}, \beta_1)$ as shown in Fig 1. $(T^*, \beta^*)$ is our desired tree decomposition with low depth, unbreakable bags and small adhesions. Recall that, $V(T^*) = V(T) \cup V(\mathcal{T})$ and $E(T^*) = \{(\tau(x), x) : x \in T\} \cup \{(\gamma(t), t) : t \in \mathcal{T} \setminus \{r_{\mathcal{T}}\}\}$.

For $t \neq r_{\mathcal{T}}$, $t \in V(\mathcal{T})$. Since $(T, \beta)$ is a tree decomposition and $E_t$ are the only set of edges in $\mathcal{T}$ between vertices in $\tau^{-1}(t)$ and $\tau^{-1}(\mathcal{P}(t))$. Further since $(\mathcal{T}, \beta_1)$ is a tree decomposition with $\beta_1(t) = \bigcup_{x \in \tau^{-1}(t)} \beta(x)$, $\beta_1(\mathcal{P}(t)) \cap \beta_1(t) \subseteq Y_t$.

Recall that $(T, \beta)$ is a tree decomposition. Therefore, for any two nodes $x, y \in V(T)$, consider a vertex $v \in \beta(x) \cap \beta(y)$. Note that $v$ must belong to every bag of the node appearing on the unique $x$ to $y$ path in $T$. Let $z \in V(T)$ be the least common ancestor of $x$ and $y$ – note that $z$ may be equal to $x$ or $y$ or neither. Suppose $z \notin \{x, y\}$. Then, $v$ must appear in $\beta(x) \cap \beta(\mathcal{P}(x)) = \sigma(x, \mathcal{P}(x))$, as well as in $\sigma(y, \mathcal{P}(y))$. Otherwise, if $z = x$ (w.l.o.g.), then $v \in \sigma(y, \mathcal{P}(y))$. Since $Y_t \supseteq \sigma(x, \mathcal{P}(x)) \cup \sigma(y, \mathcal{P}(y))$, we get the third property.

Let $t_c \in V(\mathcal{T})$, $t_c \neq r_{\mathcal{T}}$. Further let $t = \mathcal{P}(t_c)$ in $\mathcal{T}$. We now show that for all but at most one vertex $x \in \tau^{-1}(t)$, $\beta_1(t_c) \cap \beta(x) \subseteq Y_t$. If for all $x \in \tau^{-1}$, $\beta_1(t_c) \cap \beta(x) \subseteq Y_t$ then we assign $\gamma(t_c) = y$ for some $y \in \tau^{-1}$. In this case, property 4 directly holds. Otherwise there is one vertex $x \in \tau^{-1}(t)$ such that $\beta_1(t_c) \cap \beta(x)$ is not a subset of $Y_t$, then we assign $\gamma(t_c) = x$. Here too, property 4 holds.

Now we show that for all but at most one vertex $x \in \tau^{-1}(t)$, $\beta_1(t_c) \cap \beta(x) \subseteq Y_t$. Since $\mathcal{T}$ is a nice tree partition, $T[V_{t_c}]$ is a subtree of $T$, where $V_{t_c} = \bigcup_{x \in V(\mathcal{T}_{t_c})} \tau^{-1}(x)$. Next $V_{t_c}$ does not contain $r_T$ because $t_c \neq r_{\mathcal{T}}$ and $\tau(r_t) = r_{\mathcal{T}}$. Further $V_{t_c}$ is tree in the forest $T \setminus \tau^{-1}(t)$. Every vertex in $\tau^{-1}(t)$ has at most one neighbor in $V_{t_c}$ otherwise it will form a cycle. Since $T$ is a rooted tree there is at most one node $x \in \tau^{-1}(t)$ whose neighbor in $V_{t_c}$ is not an ancestor (or a parent) of $x$. Thus for all others $\beta(x) \cap \beta_1(t_c) \subseteq \sigma(x, \mathcal{P}(x)) \subseteq Y_T$. ◀

Let $\gamma : V(\mathcal{T}) \to V(T)$ and $Y_t \subseteq \beta_1(t)$ for each node $t \in \mathcal{T}$ be function and sets given by Lemma 6. We now define a pair $(T^*, \beta^*)$ based on $T, \mathcal{T}, \gamma$ and $Y_t$ that we will prove to be a tree decomposition of $G$ having all our desired properties including unbreakable bags.

▶ **Definition 7** ($(T^*, \beta^*)$). *Let $T^*$ be a graph with $V(T^*) = V(T) \cup V(\mathcal{T})$ and $E(T^*) = \{(\tau(x), x) : x \in T\} \cup \{(\gamma(t), t) : t \in \mathcal{T} \setminus \{r_{\mathcal{T}}\}\}$. Also let $\beta^* : V(T^*) \to 2^{V(G)}$ be a function with $\beta^*(t) = Y_t$, for $t \in \mathcal{T}$ and $\beta^*(x) = Y_{\tau(x)} \cup \beta(x)$ for $x \in T$.*

In the following, we show that $(T^*, \beta^*)$ is a tree decomposition of $G$ (see Figure 2).

▶ **Lemma 8** ($*$). *$(T^*, \beta^*)$ is a rooted tree decomposition of $G$. Further $(T^*, \beta^*)$ satisfies the following properties:*

1. *every adhesion of $(T^*, \beta^*)$ is of size at most $8k$*
2. *every bag of $(T^*, \beta^*)$ is $(q + 8k, k)$-unbreakable in $G$.*
3. *$T^*$ has depth at most $2\lceil \log_2 |V(G)| \rceil$*

We are now ready to prove the main theorem of this section.

▶ **Theorem 9.** *There exists a polynomial-time algorithm that takes input an $n$-vertex graph $G$ and positive integers $k$ and $q$, and a rooted tree decomposition $(T, \beta)$ of $G$ satisfying the following properties:*
1. *every adhesion of $(T, \beta)$ is of size at most $k$*
2. *every bag of $(T, \beta)$ is $(q, k)$-unbreakable in $G$*

*and finds a compact tree decomposition $(T', \beta')$ of $G$ satisfying the following properties:*
1. *every adhesion of $(T', \beta')$ is of size at most $8k$*
2. *every bag of $(T', \beta')$ is $(q + 8k, k)$-unbreakable in $G$.*
3. *$T'$ has depth at most $2\lceil \log_2 n \rceil$.*

**Proof.** Let $(T, \beta)$ be the input tree decomposition of $G$. We first compute a nice tree partition $(\mathcal{T}, \tau)$ of $T$ using Lemma 4. Then we obtain the tree decomposition $(\mathcal{T}, \beta_1)$ of $G$ where $\beta_1 : V(G) \to V(\mathcal{T})$ and $\beta_1(t) = \bigcup_{x \in \tau^{-1}(t)} \beta(x)$ – it is a tree decomposition by Lemma 5.

Let $\beta^* : V(T^*) \to 2^{V(G)}$ be a function with $\beta^*(t) = Y_t$, for $t \in \mathcal{T}$ and $\beta^*(x) = Y_{\tau(x)} \cup \beta(x)$ for $x \in T$. We compute the tree decomposition $(T^*, \beta^*)$ with $V(T^*) = V(T) \cup V(\mathcal{T})$ and $E(T^*) = \{(\tau(x), x) : x \in T\} \cup \{(\gamma(t), t) : t \in \mathcal{T} \setminus \{r_{\mathcal{T}}\}\}$. By Lemma 8 it satisfies all our required properties except compactness. We can in polynomial time obtain a compact tree decomposition $(T', \beta')$ whose each bag is a subset of some bag of $(T^*, \beta^*)$ and whose height is the same as $\mathcal{T}^*$ [5]. Thus the tree decomposition $(T', \beta')$ will satisfy all our required properties. ◀

The following corollary directly follows from Theorem 9, and a known result ([9]) that outputs a tree decomposition of a graph satisfying the premise of Corollary 10 in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$.

▶ **Corollary 10.** *Given an $n$-vertex graph $G$ and an integer $k$, one can in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ compute a rooted compact tree decomposition $(T, \beta)$ of $G$ such that:*
1. *Every adhesion of $(T, \beta)$ is of size at most $8k$*
2. *Every bag of $(T, \beta)$ is $(9k, k)$-unbreakable in $G$*
3. *$T$ has depth at most $2\lceil \log_2 n \rceil$*

## 4 Exact and Approximation algorithms

Let $(G, c, k, r^\circ, \chi)$ be an instance of Fair Bisection and let $n = |V(G)|$. We start by invoking the algorithm of Theorem 10 with $G$ and $k$ to obtain a rooted compact tree decomposition $(T, \beta)$ of $G$ with root $r$, having $(9k, k)$-edge-unbreakable bags and adhesions of size at most $8k$. This takes time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. Recall that an edge cut $(A, B)$ is $(\epsilon, r^\circ)$-*fair* if $A$ and $B$ contain no more than $r_i(1 + \epsilon)$ and $(c_i - r_i)(1 + \epsilon)$ vertices respectively.

▶ **Theorem 11.** *Given an instance $(G, c, k, r^\circ, \chi)$ of Fair Bisection and $\epsilon > 0$ there exists an algorithm that in time $2^{\mathcal{O}(k \log k)} \cdot \left(\frac{c}{\epsilon}\right)^{\mathcal{O}(c)} \cdot n^{\mathcal{O}(1)}$ finds an $(\epsilon, r^\circ)$-fair edge cut of $G$ if one exists, else returns no.*

Given a subset $S \subseteq V(G)$, we use $\chi^\circ(S)$ to denote the $c$ length tuple where the $i^{th}$ entry is the number of vertices $v$ in $S$ having color $i$, i.e. $\chi(v) = i$. We remark that we use $^\circ$ to denote tuples of integers of length $c$. Further we use operators such as $+, -$, scalar multiplication, and $\lceil\rceil$ on tuples, which perform the respective operations on each entry in the tuple(s).

For a node $t \in V(T)$ recall that $\gamma(t) = \bigcup_{s:\text{ descendant of } t} \beta(s)$, $\alpha(t) = \gamma(t) \backslash \sigma(t)$, $G_t = G[\gamma(t)] - E(G[\sigma(t)])$. We perform bottom-up dynamic programming on $(T, \beta)$. For each node $t \in V(T)$, we first define a Boolean function $f_t : \{0, \cdots, k\} \times 2^{\sigma(t)} \times \{0, \cdots, n\}^c \times \{0, \cdots, n\}^c \to \{\mathsf{True}, \mathsf{False}\}$. For each integer $w \in \{0, \cdots, k\}$, subset $A_t \subseteq \sigma(t)$, and $c$ length tuples $a^\circ$ and $b^\circ$ with $a^\circ, b^\circ \in \{0, \cdots, n\}^c$, we define the following.

▶ **Definition 12.** *$f_t(w, A_t, a^\circ, b^\circ) = \mathsf{True}$ if there exists an edge cut $(A, B)$ of $G_t$ that satisfies the following properties: (1) $(A, B)$ has order at most $w$, (2) $A \cap \sigma(t) = A_t$, (3) $\chi^\circ(A \cap \alpha(t)) = a^\circ$, and (4) $\chi^\circ(B \cap \alpha(t)) = b^\circ$. If such a cut does not exist, $f_t(w, A_t, a^\circ, b^\circ) = \mathsf{False}$. Further if $f_t(w, A_t, a^\circ, b^\circ) = \mathsf{True}$, we say an edge cut $(A, B)$ of $G_t$ that satisfies properties $1 - 4$ **realizes** $f_t(w, A_t, a^\circ, b^\circ)$.*

From the definition of $f_t$ one can make the following observation.

▶ **Observation 13.** *$(G, c, k, r^\circ, \chi)$ is a yes-instance to* Fair Bisection *if and only if for $f_r(k, \emptyset, r^\circ, c - r^\circ) = \mathsf{True}$, where $r$ is the root of $T$.*

In order to reduce the size of the domain of $f$ (and hence the running time), we work with the *reduced* domain $D = \left\{(1 + \delta)^i : i \geq 0\right\}$. This will approximate the number of vertices of each color at either side of the cut to the nearest power of $1 + \delta$, where $\delta > 0$ is a parameter whose value will be fixed later.

Let $\mathcal{C}_t$ be the set of all possible edge-cuts $(A, B)$ of $G_t$. To compute $f_t$ we have a *table* $M_t : \{0, \cdots, k\} \times 2^{\sigma(t)} \times D^c \times D^c \to \{\mathcal{C}_t \cup \perp\}$ that satisfies properties $M_t \to f_t$ and $f_t \to M_t$ (defined below in Definition 14 and 16). $M_t$ will help us to approximately obtain $f_t$. Let $z \geq 0$ be a sufficiently large constant; for example, $z = 10$ suffices. We have Definitions 14 and 16 that will be crucial towards proving the correctness of the approximation algorithm.

▶ **Definition 14** (Property $M_t \to f_t$)**.** *If $M_t(w, A_t, a^\circ, b^\circ) \neq \perp$ then $\exists\ x^\circ \in \{0, \cdots, n\}$ and $y^\circ \in \{0, \cdots, n\}$ such that:*

- *$f_t(w, A_t, x^\circ, y^\circ) = \mathsf{True}$ and $M_t(w, A_t, a^\circ, b^\circ)$ is an edge-cut that realizes $f_t(w, A_t, x^\circ, y^\circ)$*
- *$a^\circ \leq x^\circ \leq (1 + \delta)^{z \cdot \mathsf{ht}(t) \log^2 n} \cdot a^\circ$*
- *$b^\circ \leq y^\circ \leq (1 + \delta)^{z \cdot \mathsf{ht}(t) \log^2 n} \cdot b^\circ$*

▶ **Definition 15** (Global-feasible edge cut)**.** *An edge cut $(A, B)$ is global-feasible if there exists an edge cut $(A', B')$ of $G$ having order at most $k$ which induces the cut $(A, B)$ on $A \cup B$.*

▶ **Definition 16** (Property $f_t \to M_t$)**.** *If $f_t(w, A_t, x^\circ, y^\circ) = \mathsf{True}$ and there is a global-feasible edge cut $(A, B)$ of $G_t$ that realizes it then $\exists\ a^\circ \in D^c$ and $b^\circ \in D^c$ such that:*

- *$M_t(w, A_t, a^\circ, b^\circ) \neq \perp$*
- *$a^\circ \leq x^\circ \leq (1 + \delta)^{z \cdot \mathsf{ht}(t) \log^2 n} \cdot a^\circ$*
- *$b^\circ \leq y^\circ \leq (1 + \delta)^{z \cdot \mathsf{ht}(t) \log^2 n} \cdot b^\circ$*

▶ **Definition 17** (Good $M_t$)**.** *$M_t$ is* good *if it satisfies properties $M_t \to f_t$ and $f_t \to M_t$.*

▶ **Lemma 18.** *For each $\epsilon > 0$ and $\delta = \frac{\epsilon}{2z \log^3 n}$, if $f_r(k, \phi, r^\circ, c^\circ - r^\circ) = \mathsf{True}$ and $M_r$ is good, then $\exists\ a^\circ, b^\circ \in D^c$ such that $M_r(k, \phi, a^\circ, b^\circ)$ is a $(\epsilon, r^\circ)$-fair edge cut of $G$.*

**Proof.** We first note that, if $\delta \coloneqq \frac{\epsilon}{2z \log^3 n}$, then $(1+\delta)^{z \cdot \log^3 n} \leq 1+\epsilon$. This is because $\ln(1+\epsilon) \geq \frac{\epsilon}{1+\epsilon} \geq \frac{\epsilon}{2}$, since $\epsilon \in (0, 1)$, which implies that $(1 + \delta)^{z \cdot \log^3 n} \leq \exp\left(\frac{\epsilon}{2z \log^3 n} \cdot z \log^3 n\right) \leq \exp\left(\frac{\ln(1+\epsilon)}{z \log^3 n} \cdot z \log^3 n\right) = 1 + \epsilon$. Furthermore, $\log_{1+\delta} n = (\log n/\epsilon)^{\mathcal{O}(1)}$.

Let $f_r(k, \phi, r^\circ, c^\circ - r^\circ) = \mathsf{True}$ and $M_r$ satisfy properties $M_r \to f_r$ and $f_r \to M_r$. Since $ht(T) = \log n$, by the previous claim $(1+\delta)^{z \cdot \mathsf{ht}(t) \log^2 n} \leq 1 + \epsilon$. So by property $f_r \to M_r$, $\exists\ a^\circ, b^\circ \in D^c$ such that: (1) $M_r(k, \phi, a^\circ, b^\circ) \neq \perp$, (2) $a^\circ \leq r^\circ \leq (1+\epsilon) \cdot a^\circ$, and $b^\circ \leq c^\circ - r^\circ \leq (1+\epsilon) \cdot b^\circ$.

Further by property $M_r \to f_r$, since $M_r(k, \phi, a^\circ, b^\circ) \neq \perp$, $\exists\ x^\circ, y^\circ \in \{0, \cdots, n\}$ such that, (1) $f_r(k, \phi, x^\circ, y^\circ) = \mathsf{True}$ and $M_r(k, \phi, a^\circ, b^\circ)$ is an edge-cut that realizes $f_r(k, \phi, x^\circ, y^\circ)$, (2) $a^\circ \leq x^\circ \leq (1+\epsilon) \cdot a^\circ \leq (1+\epsilon) \cdot r^\circ$, and (3) $b^\circ \leq y^\circ \leq (1+\epsilon) \cdot b^\circ \leq (1+\epsilon) \cdot (r^\circ - c^\circ)$. Thus, $M_r(k, \phi, a^\circ, b^\circ)$ is a $(\epsilon, r^\circ)$-fair edge-cut of $G$. This completes our proof. ◀

Lemma 18 shows us that computing a good table $M$ efficiently is sufficient for obtaining our final approximation. We now state as a theorem that we can compute a good $M_t$ assuming a good $M_{t'}$ has been computed for each $t' \in \mathsf{child}(t)$.

▶ **Lemma 19** (∗). *There exists an algorithm that takes as input $t \in V(T)$, $\delta > 0$, $(T, \beta)$, and a good $M_{t'}$ for each $t' \in \mathsf{child}(t)$ and computes a good $M_t$ in time $2^{\mathcal{O}(k \log k)} (\log_{1+\delta} n)^{\mathcal{O}(c)} n^{\mathcal{O}(1)}$.*

Assuming Lemma 19 and Lemma 18, the correctness of our algorithm follows. Setting $\delta$ as in Lemma 18, we obtain our desired runtime thus proving Theorem 11.

**Proof of Theorem 11.** Let $\delta := \frac{\epsilon}{2z \log^3 n}$. In our algorithm we compute $M$ by computing good $M_t$ using Lemma 19 for each $t \in V(T)$, bottom up, starting from leaves of $T$ to root of $T$. We finally go over each $a^\circ, b^\circ \in D^c$ and output a cut $M_r(k, \phi, a^\circ, b^\circ)$ that is a $(\epsilon, r^\circ)$-fair edge cut of $G$ if one exists.

The correctness follows directly from the definition of $f$ and Lemma 18. The time taken by our algorithm is equal the size of domain of $M$ times the time taken to compute each entry in $M$. The size of the domain of $M$ is at most $2^{\mathcal{O}(k)} (\log_{1+\delta} n)^{\mathcal{O}(c)} n^{\mathcal{O}(1)}$ because $|D| \leq \log_{1+\delta} n$ and all adhesions in $(T, \beta)$ have size at most $8k$. The time taken to compute each entry in $M$ is $2^{\mathcal{O}(k \log k)} (\log_{1+\delta} n)^{\mathcal{O}(c)} n^{\mathcal{O}(1)}$ by Lemma 19.

Using $\log_{1+\delta} n = (\log n / \epsilon)^{\mathcal{O}(1)}$ and a standard case analysis on whether $c \leq \frac{\log n}{\log \log n}$, it follows that the total time taken is $2^{\mathcal{O}(k \log k)} (\log_{1+\delta} n)^{\mathcal{O}(c)} n^{\mathcal{O}(1)} \leq 2^{\mathcal{O}(k \log k)} \left(\frac{c}{\epsilon}\right)^{\mathcal{O}(c)} \cdot n^{\mathcal{O}(1)}$. ◀

## Computing $M_t$: A sketch of proof of Lemma 19

In particular, we design an algorithm that takes as input a graph $G$, the tree decomposition $(T, \beta)$ and a node $t$ of $T$, together with dynamic programming tables $M_{t'}$ for every child $t'$ of $t$, and outputs the appropriate dynamic programming table $M_t$ (which is good) for $t$. This algorithm is an adaptation of a similar step performed by Cygan et al. [10] in their algorithm for the Minimum Bisection problem. The algorithm of Cygan et al. [10] proceeds by a random coloring step, followed by a "knapsack"-like dynamic programming algorithm. Our algorithm proceeds in a similar manner, but faces the following key difficulty: in order to keep time and space bounded by $f(k, c, \epsilon) n^{O(1)}$ we can only store approximate values in the knapsack dynamic programming table (the table satisfies soundness and completeness properties similar to Definition 17). Therefore, after computing each entry of the table (from previous entries) we need to perform a rounding step that introduces a $(1 + (\frac{\epsilon}{\log n})^{\mathcal{O}(1)})$ multiplicative factor in the error bound. The standard way of solving Knapsack involves considering each item in the input one by one, however this would lead to the rounding error possibly accumulating and getting out of hand. We overcome this by organizing the dynamic program in a complete binary tree. That is, split the items in two equal sized groups, compute dynamic programming tables for the two groups recursively, and combine

the dynamic programming tables to the two halves to a dynamic programming for all the items. This ensures that the total error is upper bounded by a multiplicative factor of $(1+(\frac{\epsilon}{\log n})^{\mathcal{O}(1)})^{\mathcal{O}(\log^3 n)} = 1+\mathcal{O}(\epsilon)$.

## 5 Hardness

Here, we sketch the proof of our result that establishes W[1]-hardness of Fair Bisection. To this end, we first consider the following problem, called Multi-Dimensional Subset Sum. In this problem, we are given an instance $(\mathcal{V}, T)$, where $\mathcal{V} = \{V_1, \ldots, V_n\}$, such that each $V_i \in \mathcal{V}$ is a $d$-dimensional vector, i.e., $V_i \in \mathbb{Z}^d_{\geq 0}$; and $T \in \mathbb{Z}^d_{\geq 0}$ is the $d$-dimensional *target vector*. The task is to determine whether there exists a subset $U \subseteq \mathcal{V}$ such that $\sum_{V_i \in \mathcal{U}} V_i = T$?

Although it is folklore that Multi-Dimensional Subset Sum is W[1]-hard parameterized by the dimension $d$, we are unable to find a reference for this result. Thus, we give a reduction from Binary Constrainted Satisfaction Problem to MDSS; the W[1]-hardness of the former problem was established in [23, 21]. In fact, our reduction shows that MDSS is W[1]-hard even when the integer entries in each vector are bounded by a polynomial in $n$. As the first step of our reduction, given an instance $(\mathcal{V}, T)$ of MDSS, we reduce it to Multi-Dimensional Partition (MDP), where the target vector $T$ is exactly half of the sum of entries along each dimension. Now, for each vector $V_i \in \mathcal{V}$, we create a subset $U_i$ of vertices, which contains exactly $V_i(j)$ many vertices of color $1 \leq j \leq d$. Then, we arbitrarily choose a vertex in $U_i$ and connect the rest of the vertices in $U_i$ to it, making a connected component a star. Proceeding this way for each vector $V_i$, we obtain a graph $G$ that is a disjoint union of $n$ stars on $U_i$'s, with each $U_i$ containing at most polynomially many vertices of each color. It is straightforward to see the equivalence between the instance $(\mathcal{V}, T)$ of MDP, and the resulting instance of Fair Bisection, with the cut-size $k$ being zero. Thus, we conclude with the following theorem, whose formal proof can be found in the full version.

▶ **Theorem 20.** Fair Bisection *is* W[1]*-hard parameterized by the number of colors $c$, even when $k$, the cut-size is zero.*

## 6 Conclusion

In this paper, we initiated the study of Fair Bisection from the perspective of parameterized algorithms. We showed that the problem is W[1]-hard parameterized by the number of colors $c$, even when $k = 0$; thus, we cannot hope to generalize the FPT algorithm to Fair Bisection with a running time of the form $f(k, c) \cdot n^{\mathcal{O}(1)}$. On the other hand, the known $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ algorithm for Minimum Bisection ([9, 10]) extends to Fair Bisection in a straightforward manner with running time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(c)}$. Our main result is that Fair Bisection admits an FPT-approximation algorithm that finds an $(\epsilon, \mathsf{r})$-fair bisection in time $2^{\mathcal{O}(k \log k)} \cdot \left(\frac{c}{\epsilon}\right)^{\mathcal{O}(c)} \cdot n^{\mathcal{O}(1)}$. In fact, by setting $\epsilon = 1/(2n)$, we can obtain the previously mentioned exact algorithm as a corollary.

We note that our approximation algorithm also works in the setting where a vertex can belong to multiple color classes. Also, our technique can be extended to Fair $q$-section problem, where we want to partition the vertex set into $q$ parts such that (i) at most $k$ edges with endpoints in different parts, and (ii) each part has proportional representation from each color – here, the algorithm will have an XP dependence on $q$.

Our main conceptual contribution is the observation that it is possible to design parameterized approximation algorithms by applying the technique of Lampis [20] to design DP over tree decompositions with unbreakable bags. Towards this goal we designed an algorithm that

given a graph $G$ and integer $k$ computes a $(9k, k)$-unbreakable tree decomposition of $G$ with logaritmic depth and adhesions of size at most $8k$ in time $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$. We expect that this will be a useful tool for obtaining parameterized approximation algorithms for other problems by using Lampis [20]-style dynamic programming over tree decompositions with unbreakable bags.

## References


1 Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and euclidean spaces and their applications. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.23.

2 Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Kasturi R. Varadarajan. A constant approximation for colorful k-center. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 12:1–12:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.12.

3 Johannes Blömer, Christiane Lammersen, Melanie Schmidt, and Christian Sohler. Theoretical analysis of the k-means algorithm–a survey. In *Algorithm Engineering*, pages 81–116. Springer, 2016.

4 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

5 Mikolaj Bojanczyk and Michal Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 407–416. ACM, 2016. doi:10.1145/2933575.2934508.

6 Xingyu Chen, Brandon Fain, Liang Lyu, and Kamesh Munagala. Proportionally fair clustering. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1032–1041. PMLR, 2019. URL: http://proceedings.mlr.press/v97/chen19d.html.

7 Yixin Chen, Ya Zhang, and Xiang Ji. Size regularized cut for data clustering. In *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pages 211–218, 2005. URL: https://proceedings.neurips.cc/paper/2005/hash/379a7ba015d8bf1c70b8add2c287c6fa-Abstract.html.

8 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM J. Comput.*, 45(4):1171–1229, 2016.

9 Marek Cygan, Pawel Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Trans. Algorithms*, 17(1):6:1–6:30, 2021. doi:10.1145/3426738.

10 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM J. Comput.*, 48(2):417–450, 2019. doi:10.1137/140988553.

11 Jefferey Dastin. Amazon scraps secret ai recruiting tool that showed bias against women. *Reuters*, 2018. https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G.

12 Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. The (im)possibility of fairness: different value systems require different mechanisms for fair decision making. *Commun. ACM*, 64(4):136–143, 2021. doi:10.1145/3433949.

13 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

14 Mehrdad Ghadiri, Samira Samadi, and Santosh S. Vempala. Socially fair k-means clustering. In Madeleine Clare Elish, William Isaac, and Richard S. Zemel, editors, *FAccT '21: 2021 ACM Conference on Fairness, Accountability, and Transparency, Virtual Event / Toronto, Canada, March 3-10, 2021*, pages 438–448. ACM, 2021. doi:10.1145/3442188.3445906.

15 Patrick J Grother, Patrick J Grother, Mei Ngan, and K Hanaoka. *Face recognition vendor test (FRVT)*. US Department of Commerce, National Institute of Standards and Technology, 2014.

16 Rudolf Halin. Tree-partitions of infinite graphs. *Discret. Math.*, 97(1-3):203–217, 1991. doi:10.1016/0012-365X(91)90436-6.

17 Lingxiao Huang, Shaofeng H.-C. Jiang, and Nisheeth K. Vishnoi. Coresets for clustering with fairness constraints. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 7587–7598, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/810dfbbebb17302018ae903e9cb7a483-Abstract.html.

18 Xinrui Jia, Kshiteej Sheth, and Ola Svensson. Fair colorful k-center clustering. In Daniel Bienstock and Giacomo Zambelli, editors, *Integer Programming and Combinatorial Optimization – 21st International Conference, IPCO 2020, London, UK, June 8-10, 2020, Proceedings*, volume 12125 of *Lecture Notes in Computer Science*, pages 209–222. Springer, 2020. doi:10.1007/978-3-030-45771-6_17.

19 Subhash Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into $_1$. *J. ACM*, 62(1):8:1–8:39, 2015. doi:10.1145/2629614.

20 Michael Lampis. Parameterized approximation schemes using graph widths. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

21 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200. SIAM, 2020. doi:10.1137/1.9781611975994.134.

22 Yury Makarychev and Ali Vakilian. Approximation algorithms for socially fair clustering. In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, volume 134 of *Proceedings of Machine Learning Research*, pages 3246–3264. PMLR, 2021. URL: http://proceedings.mlr.press/v134/makarychev21a.html.

23 Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

24 Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

25 Evelien Otte and Ronald Rousseau. Social network analysis: a powerful strategy, also for the information sciences. *Journal of information Science*, 28(6):441–453, 2002.

26 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 255–264. ACM, 2008. doi:10.1145/1374376.1374415.

**27** Lior Rokach. A survey of clustering algorithms. In *Data mining and knowledge discovery handbook*, pages 269–298. Springer, 2009.

**28** Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k-means. In Evripidis Bampis and Nicole Megow, editors, *Approximation and Online Algorithms – 17th International Workshop, WAOA 2019, Munich, Germany, September 12-13, 2019, Revised Selected Papers*, volume 11926 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2019. doi:10.1007/978-3-030-39479-0_16.

**29** Detlef Seese. Tree-partite graphs and the complexity of algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT '85, Cottbus, GDR, September 9-13, 1985*, volume 199 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1985. doi:10.1007/BFb0028825.

**30** Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

**31** Jin-Tai Yan and Pei-Yung Hsiao. A fuzzy clustering algorithm for graph bisection. *Inf. Process. Lett.*, 52(5):259–263, 1994. doi:10.1016/0020-0190(94)00148-0.

# Improved Quantum Boosting

**Adam Izdebski**[1]
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

**Ronald de Wolf**
QuSoft, CWI and University of Amsterdam, The Netherlands

## Abstract

Boosting is a general method to convert a weak learner (which generates hypotheses that are just slightly better than random) into a strong learner (which generates hypotheses that are much better than random). Recently, Arunachalam and Maity [5] gave the first quantum improvement for boosting, by combining Freund and Schapire's AdaBoost algorithm with a quantum algorithm for approximate counting. Their booster is faster than classical boosting as a function of the VC-dimension of the weak learner's hypothesis class, but worse as a function of the quality of the weak learner. In this paper we give a substantially faster and simpler quantum boosting algorithm, based on Servedio's SmoothBoost algorithm [22].

**2012 ACM Subject Classification** Theory of computation → Quantum computation theory; Computing methodologies → Machine learning

**Keywords and phrases** Learning theory, Boosting algorithms, Quantum computing

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.64

**Funding** *Ronald de Wolf*: Partially supported by the Dutch Research Council (NWO/OCW), as part of the Quantum Software Consortium programme (project number 024.003.037), and through QuantERA ERA-NET Cofund project QuantAlgo (680-91-034).

**Acknowledgements** We thank Srinivasan Arunachalam for many helpful comments, Min-Hsiu Hsieh for sending us an updated version of [24] and answering some questions about this paper, Yassine Hamoudi for answering a question about [16], and the anonymous referees for some helpful pointers.

## 1 Introduction

### 1.1 Boosting

There has been tremendous growth in machine learning research and applications, both in practice (applying all sorts of methods on all sorts of data and seeing what works well) and in theory (computational learning theory). However, not very many ideas generated in theoretical machine learning have had a large impact on machine learning practice. One of the exceptions is *boosting*, which is a simple, general, and widely applicable method to improve the generalization error of a given learning method, i.e., to convert a *weak* learner into a *strong* learner.

The set-up here is binary classification: we are trying to predict binary labels $y$ from points $x \in \mathcal{X}$. A typical case would be $\mathcal{X} = \{0,1\}^n$. We are given $m$ labeled examples $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{-1, 1\}$ where the $x_i$s are independent and identically distributed (i.i.d.) according to some unknown distribution $\mathcal{D}$, and the binary labels are determined by some unknown *target function* $f : \mathcal{X} \to \{-1, 1\}$ that we are trying to learn, i.e., $y_i = f(x_i)$. A weak learner $\mathcal{W}$ is an algorithm that can be fed a number of examples according to a specified

[1] Work done while a student at the IILC of the University of Amsterdam.


distribution $D$ (not to be confused with the unknown data-generating distribution $\mathcal{D}$) over the $m$ examples of the given sample, and that is then promised to generate a hypothesis $h : \mathcal{X} \to \{-1, 1\}$ that is slightly better than random w.r.t. that $D$:

$$\Pr_{x\sim D}[h(x) \neq f(x)] \leq 1/2 - \gamma.$$

Here $\gamma \in (0, 1/2)$ is a small but positive number that gives the quality of the weak learner. We denote the "cost" (in time complexity or whatever measure the user likes) of one run of $\mathcal{W}$ by $W$, and use this number also as an upper bound on the number of examples (distributed according to $D$) that the weak learner uses.

A hypothesis with a generalization error that is just slightly better than random is not very useful by itself. The goal of boosting is to convert the weak learner into a strong learner, which is one that produces hypotheses not only with small empirical error (i.e., w.r.t. the uniform distribution over the $m$ examples), but even with small generalization error w.r.t. the unknown target function $f : \mathcal{X} \to \{-1, 1\}$ and the unknown distribution $\mathcal{D}$ that generated the examples:

$$\Pr_{x\sim \mathcal{D}}[h(x) \neq f(x)] \leq \varepsilon.$$

Here the desired upper bound $\varepsilon$ on the final generalization error is a parameter of the strong learner. Unsurprisingly, achieving smaller $\varepsilon$ requires a larger number of examples and larger runtime. For simplicity, in this introduction we focus on the case $\varepsilon = 1/3$ (in the body of the paper we cover the general case). Similarly, the smaller the initial advantage $\gamma$ is, the more work we will have to do find a hypothesis with small generalization error.[2]

The idea of boosting is to find a hypothesis with low empirical error by combining different runs of the weak learner on different distributions. Once we have a hypothesis with small empirical error on a sufficiently large set of examples, VC-theory implies that such a hypothesis will probably also have a small generalization error.

How can we find a hypothesis with small empirical error? Because empirical error is measured w.r.t. the uniform distribution over $\{x_1, \ldots, x_m\}$, that will be our first distribution $D^1$. We run the weak learner on $D^1$, and receive a hypothesis $h_1$ that is slightly better than random w.r.t. the uniform distribution. The next iteration then biases the distribution away from the examples that are already well-classified, by increasing the probability of misclassified examples, yielding a new distribution $D^2$. We then run the weak learner again, to generate a hypothesis $h_2$ that is slightly better than random w.r.t. this new distribution, and hence hopefully better than $h_1$ on the examples that were misclassified by $h_1$. Then we bias the distribution further towards the still-misclassified examples, and so on. The intuition here is that the distributions $D^t$ "zoom in" on the hardest examples, the ones that are most difficult to classify correctly. After some $T$ iterations, the $T$ different weak hypotheses are combined into one hypothesis $h$, typically by defining the latter as the sign of a linear combination $\sum_{t=1}^{T} \alpha_t h_t$ of the $T$ weak hypotheses $h_1, \ldots, h_T$. Surprisingly, already after a relatively small number of iterations, the resulting hypothesis will have small empirical error! Thus boosting converts the ability to generate weak hypotheses w.r.t. chosen distributions over the examples, into the ability to generate strong hypotheses, which have small error w.r.t. both the uniform distribution over the examples, and w.r.t the unknown target function $f$ and distribution $\mathcal{D}$ that generated our $m$ examples.

[2] For simplicity we will assume this $\gamma$ is known to the strong learner we are trying to design, but this is not necessary: if it doesn't know $\gamma$, the strong learner can try exponentially decreasing guesses for $\gamma$ until it finds a hypothesis with small empirical error.

A number of classical boosting algorithms exist that instantiate this meta-algorithm in different ways. The most famous of these is probably Freund and Schapire's AdaBoost [13, 14, 19] (short for "adaptive boosting"), which biases the new distribution $D^{t+1}$ based on the error $\varepsilon_t$ that $h_t$ made. It drives the empirical error all the way down to 0 (note that as soon as this error is $< 1/m$ it must actually be 0). AdaBoost uses $T = O(\log(m)/\gamma^2)$ iterations. Each iteration takes time $\tilde{O}(m)$ to compute the error $\varepsilon_t$ of $h_t$ and to update the distribution over the $m$ examples[3] and runs the weak learner $\mathcal{W}$ once, at cost $W$. This gives overall complexity

$$\tilde{O}\left(\frac{W+m}{\gamma^2}\right).$$

How large should $m$ be in order to make the inference from low empirical error to low generalization error? This depends on the hypothesis space $\mathcal{H}_{weak}$ of the weak learner, in particular on its *VC-dimension* $d$ (defined in Section 2.1). The hypothesis space $\mathcal{H}_{strong}$ of the boosting algorithm consists of all signs of linear combinations of up to $T$ elements of $\mathcal{H}_{weak}$. One can show that the VC-dimension of $\mathcal{H}_{strong}$ is $D = \tilde{O}(dT)$. VC-theory implies that (for constant $\varepsilon$) $m \approx D \approx dT \approx d/\gamma^2$ examples suffice to end up with generalization error $\leq \varepsilon$ (with high probability over the choice of the sample). Accordingly, when re-expressed as a function of $d$ rather than $m$, the complexity of AdaBoost is

$$\tilde{O}\left(\frac{W}{\gamma^2}+\frac{d}{\gamma^4}\right). \tag{1}$$

## 1.2 Quantum boosting

In the last few years there has been a surge in interest in possible ways in which *quantum* computers might help improve machine learning (see [9] for a survey of several algorithmic approaches and [6] for quantum learning theory).

Recently, Arunachalam and Maity [5] gave the first speed-up for boosting on a quantum computer. Here the given sample is still the same classical sequence of $m$ labeled examples $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{X} \times \{-1, 1\}$, but these are now stored in a quantum-accessible classical memory, which means a quantum learner can query multiple examples in superposition.

The key insight of [5] is that the error $\varepsilon_t$ of the base classifier in the $t$-th iteration of AdaBoost can be approximated faster (in time $o(m)$) using a quantum counting algorithm; this approximation is subtle because it involves both multiplicative and additive error, in different regimes for $\varepsilon_t$. Their method works not only for boosting classical weak learners, but also for boosting *quantum* weak learners. These are fed quantum examples w.r.t. the distribution $D$:

$$\sum_{i=1}^{m} \sqrt{D(x_i)}|x_i, y_i\rangle. \tag{2}$$

If the quantum weak learner $\mathcal{W}$ expects to receive $W$ such examples, the quantum booster will have to prepare $W$ copies of this state to feed into $\mathcal{W}$.[4]

[3] The notation $\tilde{O}(f)$ means $O(f \cdot \text{polylog}(f))$.

[4] The classical boosting literature [19] distinguishes "boosting by resampling" and "boosting by reweighting". Like Arunachalam and Maity [5], we follow "boosting by resampling" and explicitly prepare the $W$ quantum or classical examples (w.r.t. $D^t$) that the weak learner needs, rather than just modifying $D^t$.

The quantum version of AdaBoost of [5] uses the same number of iterations as classical AdaBoost, but improves the complexity of each iteration (at least as a function of $m$ or $d$). Their main complexity upper bound is:

$$\tilde{O}\left(\frac{W^{1.5}\sqrt{d}}{\gamma^{11}}\right). \tag{3}$$

Comparing with the complexity of classical AdaBoost Eq. (1), this gives a speed-up over classical boosting in terms of the dependence on the VC-dimension $d$ of the weak learner's hypothesis class, but at the expense of a significant deterioration in terms of the dependence on the quality of the weak learner $\gamma$ and a milder deterioration in terms of the weak learner's cost $W$.

## 1.3 Our results

In this paper we give a simpler and faster quantum boosting algorithm. Instead of AdaBoost, our starting point will be Servedio's SmoothBoost algorithm [22], which we explain in Section 3. Servedio's motivation for smooth boosting was to deal with malicious noise (at a rate that depends on $\gamma$) in the sample better than AdaBoost. However, SmoothBoost is also very suitable for "quantization" thanks to the following advantages that it has over AdaBoost:

- SmoothBoost doesn't need to calculate or approximate the error $\varepsilon_t$ of $h_t$ on the $m$ examples, which means we don't need to apply approximate quantum counting for this.
- The distributions $D^t$ that it generates are "smooth" (whence its name), in the sense that no example has probability much bigger than the uniform probability $1/m$. Generating quantum examples as in Eq. (2) is cheaper when none of the probabilities is big.
- The weights $\alpha_t$ in the final linear combination $\sum_t \alpha_t h_t$ are all equal to 1 in SmoothBoost. In contrast, AdaBoost uses $\alpha_t = \frac{1}{2}\ln((1-\varepsilon_t)/\varepsilon_t)$, hence the quantum algorithm's approximation errors in $\varepsilon_t$ lead to approximation errors in $\alpha_t$ that need to be kept under control.

In addition to exploiting these "classical" advantages in order to obtain a simpler and faster quantum booster, we also give an improved procedure to generate quantum examples over the $m$ examples from $S$. This procedure assumes access to a non-normalized version of $D^t$ and doesn't have to worry about the normalizing factor. As explained in Section 4.1, in a way quantum mechanics will take care of the proper normalization for us.[5]

We obtain the following upper bound on the complexity of our Quantum SmoothBoost:[6]

$$\tilde{O}\left(\frac{W}{\gamma^4}+\frac{\sqrt{d}}{\gamma^5}\right). \tag{4}$$

[5] We also generate the quantum examples exactly, while [5] only generate them approximately and hence has to deal with the way the errors in this process affect the other parts of their boosting procedure. Our example-generating procedure could also be used to improve the bounds of quantum AdaBoost [5], though the result won't be as efficient as our quantum SmoothBoost. Note that if we want to use Quantum SmoothBoost with a classical weak learner $\mathcal{W}$, we can just measure the $W$ quantum examples in the computational basis to obtain the $W$ classical examples distributed according to $D^t$ that such a $\mathcal{W}$ needs as input. This still gives a speed-up in terms of the desired generalization error $\varepsilon$ compared to classically generating those $W$ examples.

[6] This bound is when we aim at constant generalization error $\varepsilon = 1/3$. We also make explicit the complexity for much smaller $\varepsilon$ (see Theorem 14).

This improves over the complexity of the booster of [5] (as given in Eq. (3)) in terms of the parameters $W$ and (especially) $\gamma$. The $\gamma$-dependence is still worse than classical AdaBoost (as given in Eq. (1)), but not by large powers anymore. It is an interesting open question whether this $\gamma$-dependence can be improved further.

### 1.4 Related work

Our main sources of inspiration for this paper were the quantum AdaBoost of Arunachalam and Maity [5] and classical SmoothBoost of Servedio [22], and we have tried in this paper to combine the best elements of both.

Here we mention a number of related quantum papers. Wang et al. [24] (which preceded [5]) give a quantum speed-up for a specific subtask of AdaBoost, namely to compute the coefficients $\alpha_t = \frac{1}{2}\ln((1-\varepsilon_t)/\varepsilon_t)$ that combine given base classifiers $h_1, \ldots, h_T$ into a good hypothesis $h = sign(\sum_t \alpha_t h_t)$. These weights $\alpha_t$ are approximated more efficiently than is possible classically using a version of approximate quantum counting. This, however, assumes the base classifiers have already been generated and sidesteps the most important aspect of AdaBoost, which is to generate the $h_t$'s adaptively by running the weak learner on a distribution $D^t$ that depends on $h_1, \ldots, h_{t-1}$. The even earlier paper by Schuld and Petruccione [21] considers quantum ensembles of classifiers (rather than the linear combinations used in boosting) and runs AdaBoost as a subroutine, but does not give a quantum boosting algorithm.

AdaBoost may be viewed as an instance of the multiplicative weights update method, see for instance the presentation in [4, Section 3.6]. There have been several quantum speed-ups for multiplicative weights methods in other contexts, particularly the quantum SDP-solvers of Brandão et al. [11, 3, 10, 2], and the very recent quantum version of the Hedge algorithm of Rebentrost et al. [18]. However, none of those speed-ups for versions of multiplicative weights seems directly applicable to our boosting setting.

## 2 Preliminaries

### 2.1 PAC learning

In this section we give a brief introduction to the PAC learning framework, which provides theoretical guarantees on learnability. The textbook by Shalev-Shwartz and Ben-David [23] provides an excellent and detailed introduction to the topic of classical PAC learning.

To formally introduce the PAC learning framework, let $\mathcal{D}$ denote a probability distribution over the set of *points* $\mathcal{X}$. We want to learn an unknown *target function* $f : \mathcal{X} \to \mathcal{Y}$. We will assume here that the set of labels $\mathcal{Y}$ is just $\{-1, 1\}$, so we are dealing with binary classification. A typical situation to keep in mind is the important special case of learning Boolean functions, where $\mathcal{X} = \{0,1\}^n$, or $\mathcal{X} = \cup_{n\geq 1}\{0,1\}^n$.

Learning begins by choosing a learning algorithm (a "learner") with an associated *hypothesis class* $\mathcal{H}$ of functions $h : \mathcal{X} \to \{-1, 1\}$. This hypothesis class could be any set of functions, but good examples to keep in mind are cases where $\mathcal{X} = \{0,1\}^n$ and $\mathcal{H}$ consists of objects with bounded computational power, for instance all Boolean circuits of at most a certain size, all neural networks with a specific depth and number of nodes, or all decision trees of at most a certain depth. We will assume that each $h \in \mathcal{H}$ has a succinct description and that we can efficiently evaluate a given $h$ on a given $x \in \mathcal{X}$. For simplicity we assume such an evaluation has unit cost.

The learner is given access to a *sample* $S = ((x_1, y_1), \ldots, (x_m, y_m))$, which is the training data. The points $x_i$ are i.i.d. generated according to an unknown distribution $\mathcal{D}$ on $\mathcal{X}$, and the labels $y_i = f(x_i)$ are determined by the target function $f$ that we are trying to learn. The learner's goal is to find an $h \in \mathcal{H}$ that fits well with the given training data, in the hope that this $h$ will generalize well to points that were not part of the data, in the sense of mostly giving the same labels as the target function. The PAC learning framework is a distribution-free setting, so we would like to design a learner that works well for every $\mathcal{D}$, in the sense of outputting a hypothesis with low generalization error.

▶ **Definition 1.** *The generalization error of $h : \mathcal{X} \to \mathcal{Y}$ w.r.t. target function $f : \mathcal{X} \to \mathcal{Y}$ under distribution $\mathcal{D}$ is*

$$err(h, f, \mathcal{D}) = \Pr_{x \sim \mathcal{D}}[h(x) \neq f(x)].$$

Generalization error is often referred to as the *true* error; it is the quantity the learner is really trying to minimize over the class $\mathcal{H}$ of available hypotheses.

As the distribution $\mathcal{D}$ is anyway unknown, the generalization error of a hypothesis $h$ cannot be calculated and the learner uses the *empirical* error of a hypothesis $h$ (the fraction of the sample that $h$ mislabels) to measure its performance, as a proxy for the generalization error.

▶ **Definition 2.** *The empirical error of $h : \mathcal{X} \to \mathcal{Y}$ w.r.t. sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ is*

$$\hat{err}(h, S) = \Pr_{i \in_R [m]}[h(x_i) \neq y_i],$$

*where $i \in_R [m]$ means that $i$ is taken uniformly at random from $[m] = \{1, \ldots, m\}$.*

▶ **Definition 3.** *An $(\varepsilon, \delta)$-PAC learner for a concept class $\mathcal{C}$ with hypothesis class $\mathcal{H}$ and sample complexity $m$, is an algorithm $\mathcal{A}$ such that the following holds for all target functions $f \in \mathcal{C}$ and all distributions $\mathcal{D}$ on $\mathcal{X}$:*

- *$\mathcal{A}$ takes as input $m$ examples $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$ where the $x_i$ are i.i.d. according to $\mathcal{D}$.*
- *$\mathcal{A}$ outputs an $h \in \mathcal{H}$ which is "Probably Approximately Correct" in the sense that*

$$\Pr[err(h, f, \mathcal{D}) \leq \varepsilon] \geq 1 - \delta,$$

*where the probability is taken over the sample and over the learner's internal randomness.*

The end goal is to find a learner with small sample complexity $m$, small error probability $\delta$, and (most important of all) small generalization error $\varepsilon$. Often we will start, however, with a "weak" learner, one whose generalization error is only slightly better than random. Since we restricted to binary labels ($\mathcal{Y} = \{-1, 1\}$), generalization error $\varepsilon = 1/2$ is no better than random guessing. A weak learner is a learner that does slightly better than that:

▶ **Definition 4** (Weak learning)**.** *A $\gamma$-weak learner $\mathcal{W}$ for concept class $\mathcal{C}$ with hypothesis class $\mathcal{H}_{weak}$ is a $(1/2 - \gamma, 0)$-PAC learner. Hypotheses returned by a weak learner are called* base classifiers.

Following Servedio [22], we assume the weak learner $\mathcal{W}$ has error probability $\delta = 0$, so it always outputs a hypothesis with generalization error $\leq 1/2 - \gamma$. If instead we start with a $\mathcal{W}$ that has non-zero error probability, say $1/3$, then we can reduce this error probability

to small $\delta > 0$ as follows. Run $\mathcal{W}$ a total number of $r = \lceil \log_3(1/\delta) \rceil$ times, each time with fresh independent examples. Then, except with probability $\leq (1/3)^r \leq \delta$, at least one of the returned hypotheses $h_1, \ldots, h_r \in \mathcal{H}_{weak}$ will have generalization error $\leq 1/2 - \gamma$. However, finding (with success probability $\geq 1 - \delta$) among these $r$ hypotheses one with such low error has a cost. Deciding (with success probability $\geq 2/3$) for a given hypothesis $h$ whether it has error $\leq 1/2 - \gamma$ under a given distribution can be done by sampling $O(1/\gamma^2)$ examples according to that distribution and estimating the fraction of examples where $h$ predicts the label correctly. Searching over the $r$ hypotheses to find a good one adds a factor of $O(r)$ to the classical cost, and reducing the overall error probability from $1/3$ to $\delta$ adds another factor of $O(\log(1/\delta))$.[7]

Suppose we ideally want to run an errorless weak learner $T$ times, namely once in each of $T$ iterations. But instead we start with a weak learner with error probability $1/3$. Reducing $1/3$ to $\delta \ll 1/T$ allows us to take a union bound over all $T$ iterations, and conclude that with high probability each of the $T$ iterations produces a base classifier with generalization error $\leq 1/2 - \gamma$ (w.r.t. the distribution $D^t$ of that iteration). Because here we assumed our weak learner has no error probability from the start, we do not have to factor in the additional cost for this error reduction, but it anyway doesn't significantly affect the complexities of classical or quantum boosting (Eqs. (1) and (4) respectively).

## 2.2 How many examples suffice to ensure small generalization error?

The number of examples that are necessary and sufficient for learning is governed by the VC-dimension of the relevant hypothesis class and by the desired generalization error, as follows. A set $S \subseteq \mathcal{X}$ of $d$ points is said to be *shattered* by $\mathcal{H}$ if for each of the $2^d$ labelings $\ell : S \to \{0,1\}$, there exists an $h \in \mathcal{H}$ that agrees with $\ell$ on the points in $S$. The *VC-dimension* of a hypothesis class $\mathcal{H}$ is the size of a largest $S$ that is shattered by $\mathcal{H}$. Intuitively, if the VC-dimension of $\mathcal{H}$ is small, then it should be relatively simple to find a good hypothesis in it, i.e., one that minimizes empirical error.

The following theorem implies that for sufficiently large $m$, *every* $h \in \mathcal{H}$ has a generalization error that is only slightly worse than its empirical error. Such a result means it suffices to look for a hypothesis with small empirical error.

▶ **Theorem 5** (Theorem 2.5 in [19]). *Let $\mathcal{H}$ be a hypothesis class of finite VC-dimension $d$. Assume that a sample $S$ of size $m$ is chosen for some target function $f$, i.i.d. according to some distribution $\mathcal{D}$. Then for every $\eta > 0$ it holds that*

$$\Pr[\exists h \in \mathcal{H} : err(h, f, \mathcal{D}) > \hat{err}(h, S) + \eta] \leq 8\Big(\frac{em}{d}\Big)^d \exp\Big\{\frac{-m\eta^2}{32}\Big\}.$$

If we set $\eta = \varepsilon/2$ and

$$m = O\left(\frac{d\log(d/(\delta\varepsilon)) + \log(1/\delta)}{\varepsilon^2}\right),$$

with a sufficiently large constant in the $O(\cdot)$, then (except with probability $\delta$), each $h \in \mathcal{H}$ has a generalization error that is at most $\varepsilon/2$ bigger than its empirical error. Accordingly, if a learner now outputs any hypothesis $h \in \mathcal{H}$ whose empirical error is $\leq \varepsilon/2$, then its generalization error will be $\leq \varepsilon$, as desired.

[7] In the quantum case the $O(1/\gamma^2)$ can be replaced by $O(1/\gamma)$ using quantum approximate counting (Theorem 7 below), and the $O(r)$ can be replaced by $O(\sqrt{r})$ using Grover's algorithm [15].

## 2.3 Quantum PAC learning and helpful quantum subroutines

In order to introduce the quantum boosting algorithm in Section 4, we explain the query model that the quantum algorithm works with. We say that an algorithm has *query access* to a string $z \in Z_a^N$ over alphabet $Z_a = \{0, \ldots, a-1\}$ if it can apply a unitary $O_z$ such that

$$O_z : |i, b\rangle \mapsto |i, b \oplus z_i\rangle,$$

where $i \in \{0,1\}^{\lceil \log N \rceil}$, $b \in Z_a$, and $\oplus$ denotes addition modulo $a$. Naturally, a quantum algorithm can apply $O_z$ on a superposition of distinct inputs $i$.

In the classical setting we assumed a learner is given a sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ of $m$ labeled examples. Here points $x_i \in \mathcal{X}$ are independently drawn from an unknown distribution $\mathcal{D}$, and labeled $y_i = f(x_i)$ according to an unknown target function $f$. Such a classical sample will still be the starting point of our quantum boosting algorithm; we assume the learner has query access to the sample (viewed as a string $z \in (\mathcal{X} \times \{-1, 1\})^m$). One may think of the sample as being stored in a quantum-accessible classical memory, sometimes called QRAM, which may be queried throughout the algorithm (incl. by the weak learner). However, our setting also encompassed the case of *synthetic* data, where we would have an efficient procedure which, on input $i$, computes the example $(x_i, y_i)$.

Even though the initially given sample is classical, like Arunachalam and Maity [5] we will set up our quantum booster so that it can work to improve a classical weak learner but also to improve a *quantum* weak learner. The latter is given *quantum examples* w.r.t. distribution $D$:

$$\sum_{x \in \mathcal{X}} \sqrt{D(x)} |x, f(x)\rangle.$$

One can think of a quantum example as the coherent version of a random example $(x, f(x))$ where $x \sim D$. A quantum learner is given access to several copies of the quantum example and performs a POVM measurement, where each outcome is associated with a hypothesis $h$ in its hypothesis class. It won't matter for the purposes of this paper, but [7] proved that in the general PAC and agnostic learning settings, the required number of classical and quantum examples are the same up to contant factor.

In the case of boosting, the weak learner will be fed quantum examples w.r.t. a distribution $D$ that only has support on the $m$ given examples. Since our initially given sample is classical, our boosting algorithm will itself have to prepare the quantum examples that it wants to feed into the weak learner in each iteration, and we have to (and will) account for the cost of this.

We also assume that we can evaluate a given $h$ (in the weak learner's hypothesis class $\mathcal{H}_{weak}$) in superposition, meaning we can apply a unitary that maps $|h\rangle|x\rangle|b\rangle \mapsto |h\rangle|x\rangle|h(x) \cdot b\rangle$; here the basis states of the first space are the names of the $h \in \mathcal{H}_{weak}$, the basis states of the second space are the elements of $\mathcal{X}$, and the basis states of the third space are the labels in $\mathcal{Y} = \{-1, 1\}$.

The definitions of PAC learning and weak learning straightforwardly generalize to the quantum setting. We refer to the survey [6] for more on this model. The following basic quantum subroutines can be derived from Brassard et al. [12] (or from [1] if one wants to avoid use of the quantum Fourier transform):

▶ **Theorem 6** (Amplitude amplification). *Suppose we have an $m$-qubit unitary $U$ such that*

$$U|0^m\rangle = \sqrt{a}|\phi_0\rangle|0\rangle + \sqrt{1-a}|\phi_1\rangle|1\rangle,$$

*and we know a lower bound $a'$ on $a$. Then there exists a quantum algorithm $V$ using $O(1/\sqrt{a'})$ applications of $U$ and $U^\dagger$, and $\tilde{O}(1/\sqrt{a'})$ other gates, such that*

$$V|0^m\rangle = \sqrt{b}|\phi_0\rangle|0\rangle + \sqrt{1-b}|\phi_1\rangle|1\rangle,$$

*where $b \in [1/2, 1]$.*

▶ **Theorem 7** (Approximate counting). *Suppose we have query access to a string $z \in [0,1]^N$, with sum $s = \sum_{i=1}^N z_i \geq 1$. There exists a quantum algorithm that uses $O(\frac{1}{\varepsilon}\sqrt{N}\log(1/\delta))$ queries and $\tilde{O}(\frac{1}{\varepsilon}\sqrt{N}\log(1/\delta))$ other operations, and that outputs (except with probability $\leq \delta$) an $\tilde{s}$ such that $(1-\varepsilon)s \leq \tilde{s} \leq (1+\varepsilon)s$.*

## 3 SmoothBoost

We first consider the classical SmoothBoost algorithm of Servedio [22]. It generates only smooth distributions, in the sense that none of the examples get too much weight. In the next section we will introduce a quantum version of SmoothBoost.

We give the pseudocode of SmoothBoost in Algorithm 1. There are a few cosmetic changes compared to the pseudocode of [22] that will make it easier for us to quantize it later. The algorithm takes four inputs. First, a $\gamma$-weak learner $\mathcal{W}$ with associated hypothesis class $\mathcal{H}_{weak}$ and cost and sample complexity $W$. Second, a sample $S = ((x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{-1, 1\})^m$, for some sample size $m$ that we will choose later. Lastly, a parameter $\kappa \in (0, 1)$ which controls the empirical error of SmoothBoost and a parameter $\theta \in [0, \frac{1}{2})$ which controls the desired margin of the output hypothesis $h$. The goal of SmoothBoost is to output a hypothesis $h : \mathcal{X} \to \{-1, 1\}$ with small empirical error (and as we shall see later, for sufficiently large $m$, this $h$ will also have large generalization error). The final $h$ is going to be the sign of a sum of elements of $\mathcal{H}_{weak}$, so the strong learner's hypothesis class is larger than that of the weak learner.

The central objects in this algorithm are the vectors $M^1, M^2, \ldots, M^T \in [0, 1]^m$, which are unnormalized distributions over the $m$ examples. The distribution $D^t$ is the normalized version of $M^t$. SmoothBoost starts by initializing weights to $N_i^0 = 0$ and $M_i^1 = 1$, for all $i \in [m]$, so $D^1$ is uniform. In each iteration, Step 6 checks whether the sum of $M_i^t$ is below $\kappa m$, and if so it terminates. Otherwise it runs the weak learner on $W$ i.i.d. examples sampled (from $S$) according to $D^t$, producing a base classifier $h_t : \mathcal{X} \to \{-1, 1\}$. Step 9 updates $N^{t-1}$ to $N^t$ and $M^t$ to $M^{t+1}$. For each $i \in [m]$, $N_i^t$ is the cumulative amount by which hypotheses $h_1, \ldots, h_t$ beat the desired margin $\theta$. If $x_i$ got correctly classified by $h_t$, then it will get higher weight $N_i^t$, which results in smaller weight $M_i^{t+1}$ and smaller probability $D_i^{t+1}$ in the next round of boosting. This mechanism forces the next run of the weak learner $\mathcal{W}$ to "zoom in" (i.e., assign higher probabilities) to systematically misclassified instances. The procedure terminates if the sum of all weights $\sum_{i\in[m]} M_i^t$ gets sufficiently small, as controlled by the parameter $\kappa$.

We now state three results from Servedio [22] that show, respectively, that the intermediate distributions are smooth, that SmoothBoost terminates after a small number of iterations, and that it returns a hypothesis with low empirical error.

▷ Claim 8 (Lemma 1 of [22]). For each $1 \leq t \leq T$, it holds that $\max_{i\in m} |D_i^t| \leq \frac{1}{\kappa m}$.

Proof. This follows from the condition of Step 6: before termination we have $\sum_{i=1}^m M_i^t \geq \kappa m$ and $M_i^t \in [0, 1]$, hence $D_i^t = M_i^t / \sum_{i\in[m]} M_i^t \leq 1/\kappa m$ for all $i \in [m]$. ◁

**Algorithm 1** SmoothBoost.

**Input:** A $\gamma$-weak learner $\mathcal{W}$ with complexity $W$.
A sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{-1, 1\})^m$.
Parameters $\kappa \in (0, 1), \theta \in [0, \frac{1}{2})$.

**Output:** Hypothesis $h : \mathcal{X} \to \{-1, 1\}$.

**function** SmoothBoost($\mathcal{W}$, $S$, $\kappa$, $\theta$)
  For all $i \in [m]$ initialize: $N_i^0 \leftarrow 0$, $M_i^1 \leftarrow 1$.
  $t \leftarrow 1$
  **while** true **do**
    Compute $s = \sum_{i=1}^m M_i^t$.
    If $s < \kappa m$ then $T \leftarrow t - 1$, **return** $h \leftarrow sign(\sum_{t=1}^T h_t)$, and **terminate**.
    Prepare $W$ i.i.d. examples w.r.t. distribution $D^t = M^t / \sum_i M_i^t$ (see Footnote 8).
    Feed those $W$ examples into the weak learner $\mathcal{W}$ to obtain base classifier $h_t$.
    For all $i \in [m]$ set

$$N_i^t \leftarrow N_i^{t-1} + h_t(x_i) y_i - \theta$$

and

$$M_i^{t+1} \leftarrow \begin{cases} 1 & \text{for } N_i^t < 0 \\ (1-\gamma)^{\frac{N_i^t}{2}} & \text{for } N_i^t \geq 0 \end{cases}$$

    $t \leftarrow t + 1$.

▷ Claim 9 (Theorem 3 of [22]). If $\theta = \frac{\gamma}{2+\gamma}$ and for all $t$ it holds that $\Pr_{i \sim D^t}[h_t(x_i) \neq y_i] \leq \frac{1}{2} - \gamma$, then SmoothBoost terminates with $T < \frac{2}{\kappa \gamma^2 \sqrt{1-\gamma}}$ iterations.

Proof. [22, Lemmas 4 and 5] imply $\frac{2m}{\gamma\sqrt{1-\gamma}} > \gamma \sum_{t=1}^T \sum_{i=1}^m M_i^t$ (this is the hard part of Servedio's correctness proof of SmoothBoost). We have $\sum_{i=1}^m M_i^t \geq \kappa m$ for all $t$ until termination. Hence $\frac{2m}{\gamma\sqrt{1-\gamma}} > \gamma T \kappa m$, which implies the claim. ◁

▷ Claim 10 (Theorem 2 of [22]). After $t$ iterations of SmoothBoost, the hypothesis $h = sign(\sum_{t=1}^t h_t)$ has empirical error $\hat{err}(h) \leq \sum_{i=1}^m M_i^{t+1}/m$.

Proof. Note that $N_i^T = \sum_{t=1}^T (h_t(x_i) y_i - \theta)$. Hence if $i$ is such that $\sum_{t=1}^T h_t(x_i) y_i < \theta T$, then $N_i^T < 0$ and $M_i^{T+1} = 1$. The final hypothesis $h$ errs on the $i$th example iff $\sum_{t=1}^T h_t(x_i) y_i < 0$. We upper bound the number of $i \in [m]$ for which this happens:

$$\left| \left\{ i \mid \sum_{t=1}^T h_t(x_i) y_i < 0 \right\} \right| \leq \left| \left\{ i \mid \sum_{t=1}^T h_t(x_i) y_i < \theta T \right\} \right| = \sum_{i : \sum_{t=1}^T h_t(x_i) y_i < \theta T} M_i^{T+1} \leq \sum_{i=1}^m M_i^{T+1}.$$

◁

Since SmoothBoost terminates if $\sum_{i=1}^m M_i^{T+1} < \kappa m$, Claim 10 implies that the empirical error of the final hypothesis is $< \kappa$.

Combining the previous two claims, we see that the empirical error decreases like $O(1/(T\gamma^2))$. This contrasts with AdaBoost, where the empirical error goes down exponentially fast in $T$ and hence can be driven down to $< 1/m$ (and hence to 0) quite cheaply.

SmoothBoost does not drive the empirical error down all the way to 0, because that would require setting $\kappa < 1/m$ which implies a very large number of iterations, $T = O(m/\gamma^2)$. However, small but non-zero empirical error is good enough for our purposes, because (with sufficiently large sample size $m$) that already implies small generalization error.

We will choose $\kappa$ to be $\varepsilon/2$, which sets the above upper bound on the empirical error of the final hypothesis $h$ to half of the allowed generalization error. By the discussion following Theorem 5, if the sample size $m$ is large enough, the final hypothesis will have generalization error $\leq \varepsilon$, as desired. The required $m$ depends on the VC-dimension of the hypothesis class $\mathcal{H}_{strong}$ of SmoothBoost, which consists of signs of sums of $T$ elements of the hypothesis class $\mathcal{H}_{weak}$ of the weak learner. The VC-dimensions of these two classes are related as follows:

▷ Claim 11 (Shalev-Shwartz & Ben-David, p. 109 [23]). Let $\mathcal{H}_{weak}$ be a hypothesis class of VC-dimension $d$ and $\mathcal{H}_{strong} = \{sign(\sum_{i=1}^{T} h_i) \mid h_1, \ldots, h_T \in \mathcal{H}_{weak}\}$. Then the VC-dimension of $\mathcal{H}_{strong}$ is $D = O(Td\log(Td))$.

By Theorem 5 and the fact that $T = O(\frac{1}{\varepsilon\gamma^2})$ it thus suffices to take

$$m = O\left(\frac{D\log(D/(\delta\varepsilon)) + \log(1/\delta)}{\varepsilon^2}\right) = O\left(\frac{d\log(d/(\delta\varepsilon\gamma))^2}{\varepsilon^3\gamma^2} + \frac{\log(1/\delta)}{\varepsilon^2}\right) \tag{5}$$

examples in order to be able to infer (with success probability $\geq 1-\delta$) generalization error $\leq \varepsilon$ from empirical error $\leq \varepsilon/2$.

Finally, let us determine the complexity of SmoothBoost, in terms of the overall number of elementary operations and queries to the sample and to the $h_t$. There are $T = O(\frac{1}{\varepsilon\gamma^2})$ iterations. Each iteration involves one application of the weak learner $\mathcal{W}$, and $\tilde{O}(m)$ other operations. The weak learner needs to be fed $W$ examples sampled according to distribution $D^t$. Using rejection sampling, we can generate $W$ such examples at cost $O(W/\kappa) = O(W/\varepsilon)$.[8]

▶ **Theorem 12.** *Let $\mathcal{W}$ be a $\gamma$-weak learner of complexity $W$ for concept class $\mathcal{C}$, with hypothesis class $\mathcal{H}_{weak}$ of VC-dimension $d$. Then given $m$ examples according to Eq. (5), SmoothBoost is an $(\varepsilon,\delta)$-PAC learner for $\mathcal{C}$, with hypothesis class $\mathcal{H}_{strong}$. It runs the weak learner $O(\frac{1}{\varepsilon\gamma^2})$ times and uses*

$$\tilde{O}(T(W/\varepsilon + m)) = \tilde{O}\left(\frac{W}{\varepsilon^2\gamma^2} + \frac{m}{\varepsilon\gamma^2}\right) = \tilde{O}\left(\frac{W}{\varepsilon^2\gamma^2} + \frac{d}{\varepsilon^4\gamma^4}\right)$$

*other operations (elementary computational steps, queries to the sample, and evaluations of base classifiers).*

## 4 Quantum Smooth Boosting

In this section we introduce our quantum version of SmoothBoost. The algorithm is given query access to a quantum (or classical) weak learner $\mathcal{W}$ with sample complexity $W$, and to a sample $S$ of size $m$. The quantum weak learner needs to be fed *quantum* examples according to the distribution $D^t$ obtained by normalizing the weight-vector $M^t$. We will start with that.

[8] Specifically, Step 7 of SmoothBoost can be implemented as follows. Sample $i \in [m]$ uniformly. With probability $M_i^t$ output $(x_i, y_i)$, and otherwise repeat. Since the probability to output $(x_i, y_i)$ is proportional to $M_i^t$, the example (if we indeed output an example) is sampled according to the desired probability distribution $D^t$. Note that the probability that we output an example in one try is $\frac{1}{m}\sum_i M_i^t \geq \kappa$, because of the condition of Step 6. Hence the expected number of repetitions before we output an example is $\leq 1/\kappa$.

## 4.1 Preparing quantum examples

Here we show how we can efficiently prepare quantum examples w.r.t. the distribution $D^t$ induced by the non-normalized $M^t$, thanks to its smoothness. This may be viewed as a quantum analogue of the classical rejection sampling sketched in Footnote 8.

▶ **Theorem 13.** *Suppose we have query access to the $m$ numbers $M_1, \ldots, M_m \in [0,1]$. Let $s = \sum_{i=1}^m M_i$ be their (unknown) sum, which has a known lower bound of $\kappa m$. Define a probability distribution $D$ on $[m]$ by $D_i = M_i/s$. Then using an expected number of $O(1/\sqrt{\kappa})$ queries and $\tilde{O}(1/\sqrt{\kappa})$ other gates, we can prepare the state*

$$\sum_{i=1}^m \sqrt{D_i}|i\rangle.$$

**Proof.** Start by preparing the uniform state

$$\frac{1}{\sqrt{m}}\sum_{i=1}^m |i\rangle|0\rangle.$$

Using two queries (the second to uncompute the value $M_i$), and a few other gates to implement a conditional rotation by angle $\arcsin\left(\sqrt{M_i}\right)$, prepare

$$\frac{1}{\sqrt{m}}\sum_{i=1}^m |i\rangle(\sqrt{M_i}|0\rangle + \sqrt{1-M_i}|1\rangle).$$

The squared norm of the part of the state ending in $|0\rangle$ is $s/m \geq \kappa$. Now use $O(1/\sqrt{\kappa})$ rounds of amplitude amplification (Theorem 6) to increase that squared norm to $\geq 1/2$. This costs $O(1/\sqrt{\kappa})$ queries and $\tilde{O}(1/\sqrt{\kappa})$ other gates.

If we measure the last qubit of the resulting state, then we obtain outcome 0 with probability $\geq 1/2$ and the state collapses to the state that we want to prepare (with an extra $|0\rangle$-qubit that we can remove). Note that we know when we succeed to produce the desired state. Since the probability of success is $\geq 1/2$, the expected number of repetitions before success is $\leq 2$. ◀

Once we have produced a copy of the state

$$\sum_{i=1}^m \sqrt{D_i}|i\rangle,$$

we can easily convert this into a quantum example

$$\sum_{i=1}^m \sqrt{D_i}|x_i, y_i\rangle$$

by querying the sample $S$.

## 4.2 Quantizing SmoothBoost

The pseudocode of Quantum Smooth Boosting is given in Algorithm 2. The algorithm receives as input a weak quantum learner $\mathcal{W}$ with sample complexity $W$, and query access to a sample $S$ of $m$ examples. Additionally, it receives two parameters $\kappa, \theta$.

The algorithm looks a bit different from classical SmoothBoost because it doesn't update the $m$-dimensional vectors $N_i^t$ and $M_i^t$ explicitly anymore; the $O(m)$ that this costs is more than we are willing to spend in the quantum case. Instead, we will store the earlier base

classifiers $h_1, \ldots, h_t$. Queries to these classifiers together with queries to the sample $S$ allow us to calculate each entry $N_i^t$ and $M_i^t$ on demand in time $\tilde{O}(t)$, via the formulas of Step 9 of SmoothBoost.

The algorithm begins by initializing $N^0$, $M^1$ and by setting $t = 1$. Like in the classical case, we iterate until the sum of the weights $\sum_{i\in[m]} M_i^t$ becomes small enough. In contrast to the classical case, we do not have the time to sum these $m$ numbers exactly, so we will instead estimate the sum with small approximation error using quantum counting (Theorem 7).

**Algorithm 2** Quantum SmoothBoost.

**Input:** A $\gamma$-weak quantum learner $\mathcal{W}$ with complexity $W$.
A sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \{-1, 1\})^m$.
Parameters $\kappa \in (0, 1), \theta \in [0, \frac{1}{2})$.

**Output:** Hypothesis $h : \mathcal{X} \to \{-1, 1\}$.

1: **function** QuantumSmoothBoost($\mathcal{W}$, $S$, $\kappa$, $\theta$)
2: $t \leftarrow 1$
3: **while** true **do**
4: Compute an estimate $\tilde{s}$ of $s = \sum_{i=1}^m M_i^t$ with multiplicative error 1.1 (using Theorem 7), where the $M_i^t$ are as defined in Step 9 of SmoothBoost (and only computed on demand).
5: If $\tilde{s} < \kappa m$ then $T \leftarrow t - 1$, **return** $h \leftarrow sign(\sum_{t=1}^T h_t)$, and **terminate**.
6: Prepare $W$ copies of example $|D^t\rangle$ w.r.t. distribution $D_i^t = \frac{M_i^t}{\sum_{i\in[m]} M_i^t}$ (using Theorem 13).
7: Feed those $W$ examples into the weak learner $\mathcal{W}$ to obtain base classifier $h_t$.
8: $t \leftarrow t + 1$.

Quantum Smoothboost runs $O(TW)$ quantum subroutines that each have some error probability. By setting this error probability to be $\ll 1/TW$, the union bound implies that the probability that at least one of them will fail, is very small. The extra cost-factor $\log(TW)$ that this error-reduction incurs will be absorbed by our $\tilde{O}(\cdot)$ notation.

If we condition on the very-high-probability event that the various quantum subroutines involved all succeed, then the weights $N^t$ and $M^t$ are just equal to the weights as they would be in classical SmoothBoost with the same number of iterations. Because our approximation $\tilde{s}$ of $s$ for the stopping criterion has small multiplicative error, the smoothness of the intermediate distributions $D^t$ before termination can be marginally worse than in classical SmoothBoost (Claim 8): For each $1 \le t \le T$, it holds that $\max_{i\in m} |D_i^t| \le \frac{1.1}{\kappa m}$.

Quantum SmoothBoost terminates if $\tilde{s} < \kappa m$. Because $\tilde{s}$ might underestimate the true $s$ by at most a factor 1.1, upon termination we have $s < 1.1\kappa m$ and hence Claim 10 implies empirical error $e\hat{r}r(h) < 1.1\kappa$. We set $\kappa = \varepsilon/2.2$ in order to ensure $e\hat{r}r(h) \le \varepsilon/2$. Like before, we choose the sample size $m$ given by Eq. (5) to ensure generalization error $\le \varepsilon$.

The total number of iterations is still $O(\frac{1}{\varepsilon\gamma^2})$.[9] It remains to determine the complexity of one iteration. The most costly steps in one iteration are Steps 4 and 6. As a subroutine these will use the fact that we can compute $M_i^t$ and $N_i^t$ using $O(t) = O(T)$ calls to the earlier base classifiers and the sample $S$.

[9] There is one small change in the proof of Claim 9: since we condition on all runs of quantum counting in Step 4 giving an estimate of $\sum_i M_i^t$ up to multiplicative error 1.1, we now have $\sum_{i=1}^m M_i^t \ge \kappa m/1.1$ for all $t$ until termination.

- Step 4. The approximation of $s = \sum_{i=1}^{m} M_i^t$ up to multiplicative error 1.1 using Theorem 7 costs $\tilde{O}(T\sqrt{m})$ (for simplicity assume $\varepsilon \gg 1/m$ to ensure the condition $s \geq 1$ in Theorem 7 holds.)
- Step 6. Preparing one copy of $|D^t\rangle$ costs $\tilde{O}(T/\sqrt{\varepsilon})$ by Section 4.1 (using our setting of $\kappa = \varepsilon/2.2$), so overall this step costs $\tilde{O}(WT/\sqrt{\varepsilon})$.

Adding these costs shows that one iteration costs $\tilde{O}(T(W/\sqrt{\varepsilon}+\sqrt{m}))$. Plugging in $T = O(\frac{1}{\varepsilon\gamma^2})$, and the same sample size $m = \tilde{O}(d/\varepsilon^3\gamma^2)$ as for classical SmoothBoost (from Eq. (5)), gives our main result:

▶ **Theorem 14.** *Let $\mathcal{W}$ be a $\gamma$-weak quantum learner of complexity $W$ for concept class $\mathcal{C}$, with hypothesis class $\mathcal{H}_{weak}$ of VC-dimension $d$. Then given $m$ examples according to Eq. (5), QuantumSmoothBoost is an $(\varepsilon, \delta)$-PAC learner for $\mathcal{C}$, with hypothesis class $\mathcal{H}_{strong}$. It runs the weak learner $O(\frac{1}{\varepsilon\gamma^2})$ times and uses*

$$\tilde{O}(T^2(W/\sqrt{\varepsilon}+\sqrt{m})) = \tilde{O}\left(\frac{W}{\varepsilon^{2.5}\gamma^4} + \frac{\sqrt{m}}{\varepsilon^2\gamma^4}\right) = \tilde{O}\left(\frac{W}{\varepsilon^{2.5}\gamma^4} + \frac{\sqrt{d}}{\varepsilon^{3.5}\gamma^5}\right)$$

*other operations (elementary computational steps, queries to the sample, and evaluations of base classifiers).*

For direct comparison with the quantum boosting result of Arunachalam and Maity [5], we instantiate this by setting $\varepsilon = \delta = 1/3$, in which case the complexity of Quantum SmoothBoost is

$$\tilde{O}\left(\frac{W}{\gamma^4} + \frac{\sqrt{d}}{\gamma^5}\right).$$

This polynomially improves over the time complexity $\tilde{O}\left(\frac{W^{1.5}\sqrt{d}}{\gamma^{11}}\right)$ of the quantum version of AdaBoost of [5], in the $W$-dependence but especially in the $\gamma$-dependence.

## 5 Future work

This work leaves open many questions for future work:

- The $\gamma$-dependence of Quantum Smoothboost is still slightly worse than in classical boosting ($1/\gamma^5$ vs $1/\gamma^4$). Is there a way to improve this further, or can we prove a lower bound on the $\gamma$-dependence for every quantum boosting algorithm that has $\sqrt{d}$-dependence on the VC-dimension of the weak learner's hypothesis class?
- Our quantum version of SmoothBoost improves the cost per iteration but not the number of iterations, which remains $T = O(1/\varepsilon\gamma^2)$. Can we reduce the number of iterations by quantizing SmoothBoost differently, or by quantizing some other boosting approach? There are classical boosting-type algorithms with fewer iterations than SmoothBoost, for instance [8], but it's not clear there how to significantly reduce the cost per iteration on a quantum computer.
- Boosting has many applications in theory and practice. Can we find applications where quantum Smoothboost is particularly suitable – some problem where the weak learner has relatively large advantage $\gamma$ and large VC-dimension $d$, so that the square-root improvement in $d$ dominates the worse dependence on $1/\gamma$?
- Can we do boosting for *agnostic* learning, where the label $y$ of an example $(x, y)$ is not determined by $x$ but $(x, y)$ is jointly generated by some distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$?

- What about learning with various kinds of noise in the sample: random classification noise, or Massart noise, or Tsybakov noise, or malicious noise? Servedio [22] designed SmoothBoost motivated by its ability to deal with malicious noise on the labels of the sample: if 1/100 of the $m$ given examples have their label flipped, then a distribution that puts probability $\leq c/m$ on each $i$ only puts total probability $\leq c/100$ on the erroneous examples. Servedio used this to give a learning algorithm for linear threshold functions that is robust against small, $\gamma$-dependent amounts of malicious noise (see [17] and references therein for follow-up work). This result straightforwardly carries over to the quantum case using our Quantum SmoothBoost (we omit the details), but we don't know much more about quantum learning with malicious noise.
- What about learning functions that have a larger range than just $\{-1, 1\}$?
- Our booster relies on a VC-dimension-based analysis of boosting. However, there are non-VC-based analyses of boosting [20], which to some extent can explain why boosting avoids overfitting even when the VC-dimension of the booster's hypothesis class gets large (i.e., when $T$ gets large). We might use this to give an alternative analysis of quantum boosters.

## References


1 Scott Aaronson and Patrick Rall. Quantum approximate counting, simplified. In *Symposium on Simplicity in Algorithms*, pages 24–32, 2020. arXiv:1908.10846.

2 Joran van Apeldoorn and András Gilyén. Improvements in quantum SDP-solving with applications. In *Proceedings of 46th ICALP*, pages 99:1–99:15, 2019. arXiv:1804.05058. doi:10.4230/LIPIcs.ICALP.2019.99.

3 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-solvers: Better upper and lower bounds. *Quantum*, 4(230), 2020. Earlier version in FOCS'17. arXiv:1705.01843.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. doi:10.4086/toc.2012.v008a006.

5 Srinivasan Arunachalam and Reevu Maity. Quantum boosting. In *Proceedings of 37th International Conference on Machine Learning (ICML'20)*, pages 377–387, 2020. arXiv:2002.05056.

6 Srinivasan Arunachalam and Ronald de Wolf. Guest column: A survey of quantum learning theory. *SIGACT News*, 48(2):41–67, 2017. arXiv:1701.06806.

7 Srinivasan Arunachalam and Ronald de Wolf. Optimal quantum sample complexity of learning algorithms. *Journal of Machine Learning Research*, 19, 2018. Earlier version in CCC'17. arXiv:1607.00932.

8 Boaz Barak, Moritz Hardt, and Satyen Kale. The uniform hardcore lemma via approximate Bregman projections. In *Proceedings of 20th ACM-SIAM SODA*, pages 1193–1200, 2009.

9 Jacob Biamonte, Peter Wittek, Nicola Pancotti, P. Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671), 2017. arXiv:1611.09347.

10 Fernando G. S. L. Brandão, Amir Kalev, Tongyang Li, Cedric Yen-Yu Lin, Krysta M. Svore, and Xiaodi Wu. Quantum SDP solvers: Large speed-ups, optimality, and applications to quantum learning. In *Proceedings of 46th ICALP*, pages 27:1–27:14, 2019. arXiv:1710.02581. doi:10.4230/LIPIcs.ICALP.2019.27.

11 Fernando G. S. L. Brandão and Krysta M. Svore. Quantum speed-ups for solving semidefinite programs. In *Proceedings of 58th IEEE FOCS*, pages 415–426, 2017. arXiv:1609.05537. doi:10.1109/FOCS.2017.45.

12 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74. AMS, 2002. arXiv:quant-ph/0005055.

13 Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 5:119–139, 1997.

14 Yoav Freund, Robert E. Schapire, and Naoki Abe. A short introduction to boosting. *Journal of the Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

15 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996. quant-ph/9605043.

16 Yassine Hamoudi, Maharshi Ray, Patrick Rebentrost, Miklos Santha, Xin Wang, and Siyi Yang. Quantum algorithms for hedging and the Sparsitron. `arXiv:2002.06003`, 2020.

17 Phil M. Long and Rocco A. Servedio. Learning large-margin halfspaces with more malicious noise. In *Proceedings of NIPS*, pages 91–99, 2011.

18 Patrick Rebentrost, Yassine Hamoudi, Maharshi Ray, Xin Wang, Siyi Yang, and Miklos Santha. Quantum algorithms for hedging and the learning of ising models. *Phys. Rev. A*, 103:012418, January 2021. `doi:10.1103/PhysRevA.103.012418`.

19 Robert E. Schapire and Yoav Freund. Boosting: Foundations and algorithms. *Kybernetes*, 2013.

20 Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proceedings of 14th International Conference on Machine Learning (ICML'97)*, pages 322–330, 1997.

21 Maria Schuld and Francesco Petruccione. Quantum ensembles of quantum classifiers. *Scientific reports*, 8(1):1–12, 2018. `arXiv:1704.02146`.

22 Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003. Earlier version in COLT/EuroCOLT'01.

23 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

24 Ximing Wang, Yuechi Ma, Min-Hsiu Hsieh, and Manhong Yung. Quantum speedup in adaptive boosting of binary classification. *Science China Physics, Mechanics & Astronomy*, 64(2):220311, 2021. `arXiv:1902.00869`.

# Finding Long Directed Cycles Is Hard Even When DFVS Is Small or Girth Is Large

**Ashwin Jacob**
Ben-Gurion University of the Negev, Beer-Sheva, Israel

**Michał Włodarczyk**
Ben-Gurion University of the Negev, Beer-Sheva, Israel

**Meirav Zehavi**
Ben-Gurion University of the Negev, Beer-Sheva, Israel

## Abstract

We study the parameterized complexity of two classic problems on directed graphs: Hamiltonian Cycle and its generalization Longest Cycle. Since 2008, it is known that Hamiltonian Cycle is W[1]-hard when parameterized by *directed treewidth* [Lampis et al., ISSAC'08]. By now, the question of whether it is FPT parameterized by the *directed feedback vertex set* (DFVS) number has become a longstanding open problem. In particular, the DFVS number is the largest natural directed width measure studied in the literature. In this paper, we provide a negative answer to the question, showing that even for the DFVS number, the problem remains W[1]-hard. As a consequence, we also obtain that Longest Cycle is W[1]-hard on directed graphs when parameterized multiplicatively above girth, in contrast to the undirected case. This resolves an open question posed by Fomin et al. [ACM ToCT'21] and Gutin and Mnich [arXiv:2207.12278]. Our hardness results apply to the path versions of the problems as well. On the positive side, we show that Longest Path parameterized multiplicatively above girth belongs to the class XP.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Hamiltonian cycle, longest path, directed feedback vertex set, directed graphs, parameterized complexity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.65

**Related Version** *Full Version*: https://arxiv.org/abs/2308.06145

**Funding** European Research Council (ERC) grant titled PARAPATH.

## 1 Introduction

**Hamiltonian Cycle (Path) Parameterized by DFVS.** In Hamiltonian Cycle (Path), we are given a (directed or undirected) graph $G = (V, E)$, and the objective is to determine whether $G$ contains a simple cycle (path) of length $n = |V(G)|$, called a *Hamiltonian cycle (path)*. Hamiltonian Cycle[1] has been widely studied, from various algorithmic (see, e.g., [7, 9–12, 15, 16, 28, 29, 37]) and structural (see, e.g., the survey [50]) points of view. This problem is among the first problems known to be NP-complete [60], and it remains NP-complete on various restricted graph classes (see, e.g., [2, 20, 46]). Nevertheless, a longest path can be found easily in polynomial-time on the large class of directed acyclic graphs (DAGs) using dynamic programming.[2] Thus, it is natural to ask – can we solve Hamiltonian Cycle efficiently on wider classes of directed graphs that resemble DAGs to some extent?

[1] Throughout the following paragraphs, we refer only to the cycle variant of the problem, but the same statements also hold for the path variant.

[2] For example, see the lecture notes at https://people.csail.mit.edu/virgi/6.s078/lecture17.pdf.

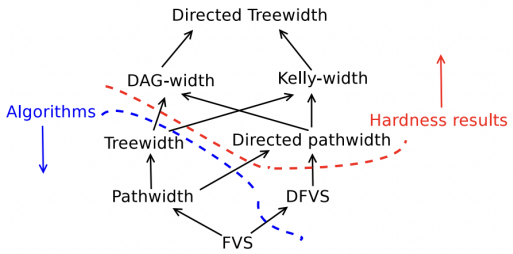



**Figure 1** Figure 3 in [59]. Caption (verbatim): *"The ecology of digraph widths. Undirected measures refer to the underlying undirected graph. Arrows denote generalizations (for example small treewidth implies small kelly-width). The dashed lines indicate rough borders of known tractability and intractability for most studied problems (including* Hamiltonian Cycle*)."*

In the undirected realm, the class of acyclic graphs (i.e., forests) can be generalized to graphs of bounded treewidth [27]. The celebrated Courcelle's theorem [24] states that a wide range of problems, including Hamiltonian Cycle, are *fixed-parameter tractable* (FPT)[3] when parameterized by treewidth. Historically, the notion of *directed treewidth* (Definition 4) was introduced by Johnson et al. [58] as a generalization of the undirected notion contrived to solving linkage problems such as Hamiltonian Cycle; see also [70]. Since then, directed treewidth has been intensively studied (see, e.g., [1, 47, 48, 52, 61, 76]). Over the years, various other width measures for directed graphs have been proposed and studied as well, where the most prominent ones are the DFVS number (which is the minimum number of vertices to remove to make the graph acyclic), *directed pathwidth* (defined similarly to directed treewidth, where we replace trees by paths), *DAG-width* [69] and *Kelly-width* [55]. Notably, DFVS is the largest among them (see Fig. 1). Johnson et al. [58] and Hunter et al. [55] proved that Hamiltonian Cycle (as well as Longest Cycle, defined later) is *slice-wise polynomial* (XP) parameterized by directed treewidth. Later, Lampis et al. [64] (originally in 2008 [63]) showed that, here, this is the "right" form of the time complexity: they proved that Hamiltonian Cycle parameterized by directed pathwidth is W[1]-hard (in fact, even W[2]-hard), implying that the problem is unlikely to be FPT.

Thus, since 2008, it has remained open whether Hamiltonian Cycle parameterized by DFVS is FPT or W[1]-hard (see Fig. 1). As a follow-up to their result, Kaouri et al. [59] wrote: *"Treewidth occupies a sweet spot in the map of width parameters: restrictive enough to be efficient and general enough to be useful. What is the right analogue which occupies a similar sweet spot (if it exists) for digraphs? One direction is to keep searching for the right width that generalizes DAGs, that is searching the area around (and probably above) DFVS. ... The other possibility is that widths which generalize DAGs, such as DFVS and all the currently known widths, may not necessarily be the right path to follow. ... The search is on!"*

Our main contribution is, essentially, the finish line for this search for Hamiltonian Cycle. We prove that, unfortunately, already for DFVS number itself, the problem is W[1]-hard. Formally, Hamiltonian Cycle (Path) By DFVS is defined as follows: Given a directed graph $G = (V, E)$ and a subset $X \subseteq V(G)$ such that $G - X$ is acyclic, determine whether $G$ contains a Hamiltonian cycle (path). Here, the parameter is $|X|$. Note that $X$ need not be given as input – the computation of a minimum-size vertex set whose removal makes a given directed graph acyclic is in FPT [23], so, given a directed graph $G$, we can simply run the corresponding algorithm to attain $X$ whose size is the DFVS number of $G$.

[3] A problem is FPT (resp. XP) if it is solvable in time $f(k) \cdot n^{O(1)}$ (resp. $n^{f(k)}$) for some computable function $f$ of the parameter $k$, where $n$ is the size of the input.

▶ **Theorem 1.** Hamiltonian Cycle By DFVS *is* W[1]*-hard.*

We also obtain the same hardness result for Hamiltonian Path By DFVS. Notice that Hamiltonian Cycle (Path) By DFVS is in XP, since, as mentioned earlier, it is already known to be in XP even when parameterized by the smaller parameter directed treewidth [58]. So, the classification of this problem is resolved.

Lastly, we remark that the choice of the larger directed feedback arc set (DFAS) number is also futile, as it also yields W[1]-hardness. Indeed, there is a simple reduction that shows this: Replace each vertex $v$ in $S$ by two vertices, $v_{\mathsf{in}}$ and $v_{\mathsf{out}}$, so that all vertices that were in-neighbors (out-neighbors) of $v$ become in-neighbors (out-neighbors) of $v_{\mathsf{in}}$ ($v_{\mathsf{out}}$), and add the arc $(v_{\mathsf{in}}, v_{\mathsf{out}})$. It is immediate that the original instance is a Yes-instance of Hamiltonian Cycle if and only if the new instance is, and that the DFVS number of the original graph equals the DFAS number of the new graph.

**Longest Cycle (Path) Above Girth.** In the Longest Cycle (Path) problem, also known as $k$-Long Cycle ($k$-Path), we are given a (directed or undirected) graph $G$ and a non-negative integer $k$, and the objective is to determine whether $G$ contains a simple cycle (path) of length at least $k$. Clearly, Longest Cycle is a generalization of Hamiltonian Cycle as the latter is the special case of the former when $k = n$. Over the past four decades, Longest Cycle and Longest Path have been intensively studied from the viewpoint of parameterized complexity. There has been a long race to achieve the best known running time on general (directed and undirected) graphs for the parameter is $k$ [5, 14, 17, 19, 22, 40, 44, 54, 62, 67, 72, 74, 77–79], where the current winners for directed graphs have time complexity $4^k \cdot n^{O(1)}$ [79] for Longest Cycle and $2^k \cdot n^{O(1)}$ [77] for Longest Path. Moreover, the parameterized complexity of the problems was analyzed with respect to structural parameterizations (see, e.g., [7, 28, 29, 33, 37, 45, 49]), on special graph classes (see, e.g., [32, 41, 42, 66, 68, 80]), and when counting replaces decision [3, 4, 6, 13, 18, 25, 26, 34, 65, 75]. In fact, Longest Path is widely considered to be one of the most well studied in the field, being, perhaps, second only to Vertex Cover [27].

We consider the "multiplicative above guarantee" parameterization for Longest Path where the guarantee is girth[4], called Longest Cycle (Path) Above Girth, which is defined as follows (for directed graphs): Given a directed graph $G = (V, E)$ with girth $g$ and a positive integer $k$, determine whether $G$ contains a cycle (path) of length at least $g \cdot k$. Here, the parameter is $k$. This parameterization of Longest Cycle (Path) was considered by Fomin et al. [36], who proved that on undirected graphs the problem is in FPT. They posed the following open question: *"For problems on directed graphs, parameterization multiplicatively above girth (which is now the length of a shortest directed cycle) may also make sense. For example, what is the parameterized complexity of* Directed Long Cycle *under this parameterization?"* This question was also posed by Gutin et al. [51] as Open Question 9. As our second contribution, we resolve this question – in sharp contrast to the undirected case, the answer is negative:

▶ **Theorem 2.** Longest Cycle Above Girth *is* W[1]*-hard.*

Again, we can transfer the hardness also to the longest path setting. On the positive side, we give a deterministic XP algorithm for the path variant of the problem.

▶ **Theorem 3.** Longest Path Above Girth *is in* XP*.*

[4] The girth of a graph is the length of its shortest cycle, and it is easily computable in polynomial time.

Therefore, the classification of this problem is resolved as well. Notice that this theorem also easily implies that when we ask for a path of length at least $g + k$, the problem is in FPT (in fact, solvable in time $2^{O(k)} \cdot n^{O(1)}$) as follows. If $k \geq g$, then we can simply run some known $2^{O(k)} \cdot n^{O(1)}$-time algorithm for Longest Path parameterized by the solution size. Otherwise, when $k < g$, we create $n$ instances of Longest Path Above Girth with parameter 2 (thus, solvable in polynomial time by our theorem), one for each vertex $v$: Replace $v$ by a path of length $g - k$ whose start vertex has all in-neighbors of $v$ as its in-neighbors and whose end vertex has all out-neighbors of $v$ as its out-neighbors, and take the disjoint union of the resulting graph and a cycle of length $g$; it is easy to see that we should return Yes if and only if the answer to at least one of these $n$ instances is Yes.

Other above/below guarantee versions of Longest Cycle (Path) were also considered in the literature; see [8,35,38,39,51,53,56,57]. Remarkably, while the parameterized complexity of all of these versions is quite well understood for undirected graphs, the resolution of the parameterized complexity of most of these versions has been (sometimes repeatedly) asked as an open question for directed graphs, where only little is known. It is conceivable that our contributions will shed light on these open questions as well in future works.

## 1.1 Techniques

**Hamiltonian Cycle By DFVS.** Before describing the ideas behind our reduction, we provide some background about the related Disjoint Paths problem. Here, we are given a graph with $k$ vertex pairs $(s_i, t_i)$ and the goal is to determine whether there exists $k$ vertex-disjoint[5] paths such that the $i$-th path connects $s_i$ to $t_i$. While Disjoint Paths is famously known to be FPT (parameterized by $k$) on undirected graphs [71], the problem becomes NP-hard on directed graphs already for $k = 2$ [43]. What is interesting for us is that when the input is restricted to be a DAG, then Disjoint Paths can be solved by an XP algorithm [43], but it is unlikely to admit an FPT algorithm as the problem is W[1]-hard [73].

The latter result suggests a starting point for extending the hardness to our problem. A simple idea to design a reduction from Disjoint Paths on DAGs is to just insert edges $t_1 \to s_2, t_2 \to s_3, \ldots, t_k \to s_1$. Then the set $\{s_1, \ldots, s_k\}$ becomes a DFVS and the existence of $k$ disjoint $(s_i, t_i)$-paths implies the existence of a long cycle. There is a catch, though. This construction might turn a No-instance into an instance with a long cycle because the cycle might traverse the vertices $s_i, t_i$ in a different order, corresponding to a family of disjoint paths which does not form a solution to the original instance. To circumvent this, we open the black box and give a reduction directly from the basic W[1]-complete problem – Multicolored Clique – while adapting some ideas from the hardness proof for Disjoint Paths on DAGs by Slivkins [73].

The construction by Slivkins uses two kinds of gadgets. First, for each $i = 1, \ldots, k$ one needs a *choice gadget* comprising two long directed paths with some arcs from the first path to the second one. The solution path corresponding to the $i$-th choice gadget must choose one of these arcs to "change the lane"; the location of this change encodes the choice of the $i$-th vertex in the clique. Next, for each pair $(i, j)$, where $1 \leq i < j \leq k$, one needs a *verification gadget* to check whether the $i$-th and $j$-th chosen vertices are adjacent. The corresponding solution path must traverse the $i$-th and $j$-th choice gadgets around the locations of their transitions. The arcs between the choice gadgets are placed in such a way that both these locations can be visited only when the corresponding edge exists in the original graph.

[5] Another studied variant involves finding edge-disjoint paths. While these two problems behave differently in some settings, this is not the case in our context.

Our reduction relies on a similar mechanism of choice gadgets, formed by $k$ directed paths, corresponding to $k$ colors in the Multicolored Clique instance. The $i$-th path is divided into blocks corresponding to the vertices of color $i$. We enforce that a Hamiltonian cycle must omit exactly one block when following such a path, thus encoding a choice of $k$ vertices. The arcs between blocks from different paths encode the adjacency matrix of the original graph. We would like to guarantee that any Hamiltonian cycle $C$ visits each omitted block $k-1$ times, utilizing the arcs mentioned above. To ensure this, we need to allow $C$ to "make a step backward" on the directed path during such a visit. On the other hand, we cannot afford to create too many disjoint cycles because DFVS should have size bounded in terms of $k$. As a remedy, we attach $k-1$ long cycles to each choice gadget, which are connected to every block in a certain way (see Figure 3 on page 8). Using the fact that every vertex in the gadget must be visited by $C$ at some point, we prove that each long cycle is entered exactly once and each choice gadget is entered exactly $k$ times in total, imposing a rigid structure on a solution. As a result, we get rid of the flaw occurring in the naive construction and obtain an equivalent instance with DFVS number $\mathcal{O}(k^2)$.

**Longest Cycle (Path) Above Girth.** Having established our main hardness result, it is easy to extend it to Longest Cycle (Path) Above Girth. We replace each vertex $v$ in the DFVS by two vertices $v_{in}$ and $v_{out}$, splitting the in- and out-going arcs of $v$ between them, and insert a long path from $v_{in}$ to $v_{out}$. This path is set sufficiently long to make the girth $g$ of the graph comparable to its size. Consequently, the original graph $G$ is Hamiltonian if and only if the second one has a cycle of length $g \cdot (\mathsf{dfvs}(G)+1)$. So, this problem is also W[1]-hard.

To design an XP algorithm for Longest Path Above Girth, we follow the win-win approach employed by the FPT algorithm for the undirected case [36], which relies on a certain version of the Grid Minor Theorem. In general, the theorem states that either a graph contains a $k \times k$-grid as a minor, or its treewidth is bounded by $f(k)$, for some (in fact, polynomial) function $f$. In the first scenario, a sufficiently long path always exists, whereas in the second scenario, the problem is solved via dynamic programming on the tree decomposition.

We take advantage of the directed counterpart of the Grid Minor Theorem: either a digraph contains a subgraph isomorphic to a subdivision of the order-$\mathcal{O}(k)$ cylindrical wall or its directed treewidth is bounded by $f(k)$, for some function $f$ [61]. We prove that in the first scenario again a sufficiently long path always exists (so we obtain a Yes-instance), while in the second scenario we can utilize the known algorithm to compute the longest path in XP time with the help of the directed tree decomposition. Finally, let us remark that this argument does not extend to Longest Cycle Above Girth because we cannot guarantee the existence of a sufficiently long cycle in a subdivision of a cylindrical wall. We leave it open whether this problem belongs to XP as well.

## 2 Preliminaries

**General Notation.** For an integer $r$, let $[r] = \{1, \ldots, r\}$, and for integers $r_1, r_2$, let $[r_1, r_2] = \{r_1, r_1+1, \ldots, r_2\}$. We refer to [27] for standard definitions in Parameterized Complexity.

**Directed Graphs.** We use standard graph theoretic terminology from Diestel's book [31]. We work with simple directed graphs (digraphs) where the edges are given by a set $E$ of ordered pairs of vertices. For an edge $e = (u, v)$ in a digraph $G$, we say that $u$ is an *in-neighbor* of $v$ and $v$ is an *out-neighbor* of $u$. We refer to $e$ as an *incoming edge of* $v$ and an *outgoing* edge of $u$. For a vertex set $S \subseteq V(G)$ we define $\partial^{out}(S) = \{(u, v) \in E(G) \colon u \in S\}$,

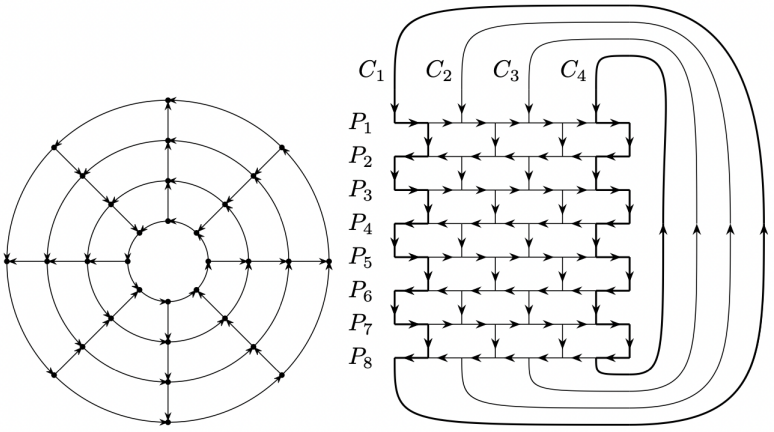



**Figure 2** A cylindrical grid and a cylindrical wall of order 4. The figure is taken from [61].

$\partial^{in}(S) = \{(u,v) \in E(G) \colon v \in S\}$, and $\partial(S) = \partial^{out}(S) \cup \partial^{in}(S)$. For two vertex sets $A, B \subseteq V(G)$ we write $E(A,B)$ for $\partial(A) \cap \partial(B)$, that is, the set of edges with one endpoint in $A$ and the other in $B$. We use this notation only when the digraph $G$ is clear from the context. When $C$ is a subgraph of $G$ we abbreviate $\partial(C) = \partial(V(C))$ and likewise for the remaining notation. A digraph $G$ is *isomorphic* to a digraph $H$ if there is a bijection $f : V(G) \to V(H)$ such that for any $u, v \in V(G)$ we have $(u,v) \in E(G)$ if and only if $(f(u), f(v)) \in E(H)$. The *length* of a path (or a cycle) $P$ is the number of edges in $P$. When the first and last vertices of a path $P$ are $s$ and $t$, respectively, we call it an *$(s,t)$-path*. A cycle in $G$ is called *Hamiltonian* if it visits all the vertices of $G$. The *girth* of a digraph $G$ is the shortest length of a cycle in $G$. For a rooted tree $T$ and a node $t \in V(T)$, $T_t$ denotes the subtree of $T$ rooted at $t$. By orienting each edge in a rooted tree from a parent to its child we obtain an *arborescence*.

**Directed Treewidth.** We move on to the directed counterparts of treewidth and grids.

▶ **Definition 4** (Directed Tree Decomposition [61]). *A* directed tree decomposition *of a directed graph $G$ is a triple $(T, \beta, \gamma)$, where $T$ is an arborescence, and $\beta : V(T) \to 2^{V(G)}$ and $\gamma : E(T) \to 2^{V(G)}$ are functions such that*

1. *$\{\beta(t) : t \in V(T)\}$ is a partition of $V(G)$ into (possibly empty) sets, and*
2. *if $e = (s,t) \in E(T)$, $A = \bigcup\{\beta(t') : t' \in V(T_t)\}$ and $B = V(G) \setminus A$, then there is no closed (directed) walk in $G - \gamma(e)$ containing a vertex in $A$ and a vertex in $B$.*

*For $t \in V(T)$, define $\Gamma(t) := \beta(t) \cup \bigcup\{\gamma(e) : e \text{ is incident with } t\}$. Moreover, define $\beta(T) := \bigcup\{\beta(t') : t' \in V(T)\}$.*

*The* width *of $(T, \beta, \gamma)$ is the least integer $w$ such that $|\Gamma(t)| \leq w+1$ for all $t \in V(T)$. The* directed treewidth *of $G$ is the least integer $w$ such that $G$ has a directed tree decomposition of width $w$.*

▶ **Definition 5** (Subdivision). *For a directed graph $G$, the* edge subdivision *of $(u,v) \in E(G)$ is the operation that removes $(u,v)$ from $G$ and inserts two edges $(u,w)$ and $(w,v)$ with the new vertex $w$. A graph derived from $G$ by a sequence of edge subdivisions is a* subdivision *of $G$.*

We now define special graphs called cylindrical grids and cylindrical walls (see Fig. 2).

▶ **Definition 6** (Cylindrical Grid and Cylindrical Wall [21]). *A* cylindrical grid of order $k$, *for some $k \geq 1$, is the directed graph $G_k$ consisting of $k$ pairwise vertex disjoint directed cycles $C_1, \ldots, C_k$, together with $2k$ pairwise vertex disjoint directed paths $P_1 \ldots, P_{2k}$ such that:*

- *for* $i \in [k]$, $V(C_i) = \{v_{i,1}, v_{i,2}, \ldots, v_{i,2k}\}$ *and* $E(C_i) = \{(v_{i,j}, v_{i,j+1}) |\ j \in [2k-1]\} \cup \{(v_{i,2k}, v_{i,1})\}$
- *for* $i \in \{1, 3, 5, \ldots, 2k-1\}$, $E(P_i) = \{(v_{1,i}, v_{2,i}), (v_{2,i}, v_{3,i}), \ldots, (v_{k-1,i}, v_{k,i})\}$, *and*
- *for* $i \in \{2, 4, 6, \ldots, 2k\}$, $E(P_i) = \{(v_{k,i}, v_{k-1,i}), (v_{k-1,i}, v_{k-2,i}), \ldots, (v_{2,i}, v_{1,i})\}$.

*A* cylindrical wall of order $k$ *is the directed graph* $W_k$ *obtained from the cylindrical grid* $G_k$ *by splitting each vertex of degree* 4 *as follows: we replace* $v$ *by two new vertices* $v_{in}, v_{out}$ *and an edge* $(v_{in}, v_{out})$ *so that every edge* $(u, v) \in E(G_k)$ *is replaced by an edge* $(u, v_{in})$ *and every edge* $(v, u) \in E(G_k)$ *is replaced by an edge* $(v_{out}, u)$.

Note that in the cylindrical grid $G_k$, the path $P_i$ is oriented from the first cycle to the last one if $i$ is odd, and from the last cycle to the first if $i$ is even (see Figure 2). We have the following theorem for directed graphs, which will be helpful in designing our algorithm in Section 4 using a win/win approach.

▶ **Theorem 7** (Directed Grid Theorem [61]). *There is a function* $f : \mathbb{N} \to \mathbb{N}$ *such that for every fixed* $k \in \mathbb{N}$*, when given a directed graph* $G$*, in polynomial time we can compute either:*

1. *a subgraph of* $G$ *that is isomorphic to a subdivision*[6] *of* $W_k$ *or*
2. *a directed tree decomposition of* $G$ *of width at most* $f(k)$.

## 3 Hardness of Hamiltonian Cycle By DFVS

This section is devoted to the proof of Theorem 1. It is based on a parameterized reduction from Multicolored Clique, defined as follows.

Multicolored Clique
**Input:** A graph $G = (V, E)$, an integer $k$, and a partition $(V^1, V^2, \ldots, V^k)$ of $V$.
**Parameter:** $k$
**Question:** Is there a clique of size $k$ with a vertex from each $V^i$, $i \in [k]$?

This problem is well-known to be W[1]-hard [27, Theorem 13.7]. For an instance $(G, (V^1, V^2, \ldots, V^k))$ of Multicolored Clique, we construct an instance $(G', X)$ of Hamiltonian Cycle By DFVS as follows.

**Construction of $G'$.** For $i \in [k]$ we construct a directed path $P^i$ corresponding to $V^i$ as follows. Let us fix an arbitrary ordering $<_i$ of the vertices in $V^i$, and accordingly, denote $V^i = \{v_1, v_2, \ldots v_{|V_i|}\}$. To each vertex $v \in V^i$ we associate a directed path $P_v$ on $2k$ vertices. We let $v^{left}$ and $v^{right}$ denote the first and last vertices of $P_v$, respectively. We refer to the $2(k-1)$ internal vertices of $P_v$ as $v^{1,out}, v^{1,in}, v^{2,out}, \ldots, v^{i-1,out}, v^{i-1,in}, v^{i+1,out}, v^{i+1,in}, \ldots, v^{k,out}, v^{k,in}$ (note that the index $i$ is avoided). The directed path $P^i$ is the concatenation of these paths: $P_{v_1} \to P_{v_2} \to, \ldots, \to P_{v_{|V^i|}}$.

For every $i \in [k]$ we create a "universal" vertex $u^i$ and insert edges $(u_i, v^{left})$, $(u_i, v^{right})$, $(v^{left}, u_i)$, $(v^{right}, u^i)$ for all $v \in V^i$. Next, we create $k-1$ cycles $C^{i \to j}$ for $j \in [k] \setminus \{i\}$, each of length $2 \cdot |V^i|$. The vertices of $C^{i \to j}$ are $c^{i \to j,out}_{v_1}, c^{i \to j,in}_{v_1}, c^{i \to j,out}_{v_2}, c^{i \to j,in}_{v_2}, \ldots, c^{i \to j,out}_{v_{|V_i|}}, c^{i \to j,in}_{v_{|V_i|}}$. We insert edges from $c^{i \to j,out}_v$ to $v^{j,out}$ and from $v^{j,in}$ to $c^{i \to j,in}_v$ for all $v \in V^i$. See Figure 3 for an illustration.

[6] The theorem in [61] states that the cylindrical wall of order $k$ is obtained as a topological minor of $G$. For any topological minor $H$ of $G$, there exists a subdivision of $H$ isomorphic to a subgraph of $G$.

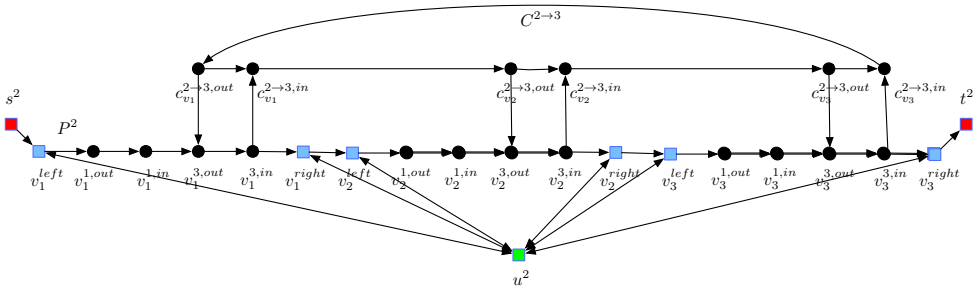



**Figure 3** Fragment of the graph $G'$: the path $P^2$ (comprising subpaths $P_{v_1}, P_{v_2}, P_{v_3}$) and the cycle $C^{2\to3}$.

We add two sets of $k$ "terminal" vertices: $S = \{s^1, s^2, \ldots, s^k\}$ and $T = \{t^1, t^2, \ldots, t^k\}$. For $i \in [k]$ we insert edges from $s^i, i \in [k]$, to $v_1^{left}$ (being the first vertex of the path $P^i$). We also add edges from $v_{|V^i|}^{right}$ (being the last vertex of the path $P^i$) to $t^i$.

We also create two additional sets of $\binom{k}{2}$ terminal vertices each: $\widehat{S} = \{\widehat{s}^{i\to j} : 1 \le 1 < j \le k\}$ and $\widehat{T} = \{\widehat{t}^{i\to j} : 1 \le 1 < j \le k\}$. For $i, j \in [k], i < j$, and $v \in V^i$, we insert an edge from $\widehat{s}^{i\to j}$ to $v^{j,\mathsf{in}}$ (which belongs to $P_v$ and hence to $P^i$) and for $v \in V^j$, we insert an edge from $v^{i,\mathsf{out}}$ to $\widehat{t}^{i\to j}$. Furthermore, we add edges from every $t \in T \cup \widehat{T}$ to every $s \in S \cup \widehat{S}$.

Finally we encode the adjacency matrix of $G$: for each edge $uv$ in $E(G)$ with $u \in V^i$ and $v \in V^j$ with $i < j$, we insert an edge from $u^{j,\mathsf{out}}$ to $v^{i,\mathsf{in}}$. See Figure 4 for an example.

This concludes the construction of $G'$. Clearly, the graph $G'$ can be computed in polynomial time when given $(G, (V^1, \ldots, V^k))$. We begin the analysis of $G'$ by showing that it has a DFVS of size $\mathcal{O}(k^2)$.

▶ **Lemma 8.** *There exists a subset $X \subseteq V(G')$ such that $G' - X$ is a DAG and $|X| = k(k-1) + 2k + \binom{k}{2}$.*

**Proof.** Let $Y$ be the set of vertices $\{c_{v^i}^{i\to j,\mathsf{out}} : i, j \in [k], i \ne j\}$ where $v^i$ stands for the first vertex in $V^i$. We set $X = Y \cup T \cup \widehat{T} \cup \{u^1, \ldots u^k\}$ and claim that $G' - X$ is a directed acyclic graph. First observe that for each $i \in [k]$ the graph $L^i := P^i \cup \bigcup_{j\ne i} C^{i\to j} - X$ is acyclic because it can be drawn with each edge being either vertical or horizontal facing right (see Figure 3). The remaining edges in $G' - X$ either start at $S \cup \widehat{S}$ (these vertices have no incoming edges) or go from $L^i$ to $L^j$ for $i < j$. Thus $G' - X$ is a DAG. Finally we check that $|X| = |Y| + |T| + |\widehat{T}| + k = k(k-1) + 2k + \binom{k}{2}$. ◀

**Correctness.** We first show that if $(G, (V^1, V^2, \ldots, V^k))$ is a Yes-instance then there exists a Hamiltonian cycle in $G'$. In the following lemmas, we refer to the directed feedback vertex set $X$ from Lemma 8.

▶ **Lemma 9.** *If $(G, (V^1, V^2, \ldots, V^k))$ is a Yes-instance of* Multicolored Clique *then $(G', X)$ is a Yes-instance of* Hamiltonian Cycle By DFVS.

**Proof.** Let $\{v_1, \ldots, v_k\}$ be a clique in $G$ with $v_i \in V^i$ for $i \in k$. For $i \in [k]$ we define the path $Z^i$ that starts at $s^i$, follows $P^i$ until $v_i^{left}$, visits $u^i$, and then follows $P^i$ from $v_i^{right}$ to $t^i$. Next, for $i < j$ we construct the path $Q^{i\to j}$ as $\widehat{s}^{i\to j} \to v_i^{j,\mathsf{in}} \to C^{i\to j} \to v_i^{j,\mathsf{out}} \to v_j^{i,\mathsf{in}} \to C^{j\to i} \to v_j^{i,\mathsf{out}} \to \widehat{t}^{i\to j}$. Note that one can traverse the entire cycle $C^{i\to j}$ after entering it from $v_i^{j,\mathsf{in}}$ and leave towards $v_i^{j,\mathsf{out}}$. The edge $(v_i^{j,\mathsf{out}}, v_j^{i,\mathsf{in}})$ is present in $G'$ due to the encoding of the adjacency matrix of $G$. The union of all these paths covers the entire vertex set of $G'$. It suffices to observe that these paths can be combined into a single cycle using the edges from $T \cup \widehat{T}$ to $S \cup \widehat{S}$. ◀

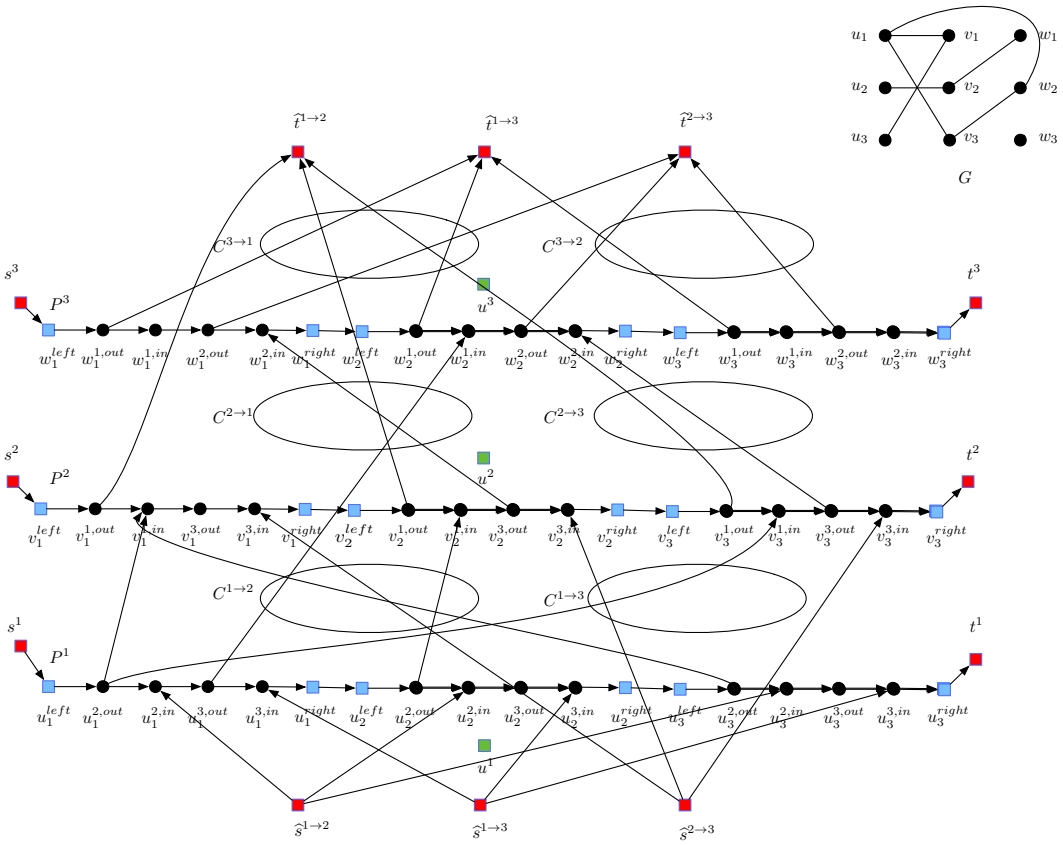



**Figure 4** Graph $G$ (top right) with a clique $\{u_1, v_3, w_2\}$ and the graph $G'$ encoding $G$. For the sake of legibility, several groups of edges are omitted in this picture: edges incident to $u^i$ or the cycles $C^{i\to j}$ and edges going from $T \cup \widehat{T}$ to $S \cup \widehat{S}$.

Proving the converse of Lemma 9 is more challenging. We will reveal the structure of a potential Hamiltonian cycle $H$ in $G'$ gradually. First, we show that if $H$ enters the cycle $C^{i\to j}$ through an incoming edge corresponding to a certain vertex $v \in V^i$, then it must also leave it through an outgoing edge related to $v$, and vice versa. A priori, there might be multiple vertices $v \in V^i$ for which this happens.

For $v \in V^i$ we denote the edge from $c_v^{i\to j,out}$ to $v^{j,out}$ by $e_v^{i\to j,out}$. Similarly, let the edge from $v^{j,in}$ to $c_v^{i\to j,in}$ be $e_v^{i\to j,in}$.

▶ **Lemma 10.** *Let $H$ be a Hamiltonian cycle in $G'$ and $1 \le i < j \le k$. Then $E(H) \cap \partial(C^{i\to j})$ is of the form $\bigcup_{v\in U}\{e_v^{i\to j,out}, e_v^{i\to j,in}\}$ where $U$ is some subset of $V^i$.*

**Proof.** Note that $\partial(C^{i\to j}) = \{e_{v_1}^{i\to j,out}, e_{v_1}^{i\to j,in}, e_{v_2}^{i\to j,out}, e_{v_2}^{i\to j,in}, \dots e_{v_{|V_i|}}^{i\to j,out}, e_{v_{|V_i|}}^{i\to j,in}\}$. First suppose that for some $v \in V^i$, we have $e_v^{i\to j,in} \in E(H)$. Since $H$ is a Hamiltonian cycle in $G'$, it has to visit the vertex $c_v^{i\to j,out}$. Thus $H$ contains an outgoing edge of $c_v^{i\to j,out}$. The only two outgoing edges of $c_v^{i\to j,out}$ are $e_v^{i\to j,out}$ and $(c_v^{i\to j,out}, c_v^{i\to j,in})$. Since $e_v^{i\to j,in} \in E(H)$ is an incoming edge of $c_v^{i\to j,in}$, the edge $(c_v^{i\to j,out}, c_v^{i\to j,in})$ cannot belong to $E(H)$. The only other candidate for the outgoing edge of $c_v^{i\to j,out}$ is $e_v^{i\to j,out}$, so $e_v^{i\to j,out} \in E(H)$.

The proof of implication in the other direction is analogous. Hence for each $v \in V^i$ we have $e_v^{i\to j,out} \in E(H) \Leftrightarrow e_v^{i\to j,in} \in E(H)$. ◀

As the next step, we show that when $v \in V^i$ and a Hamiltonian cycle enters some cycle $C^{i\to j}$ adjacent to $P^i$ through a vertex from $P_v$, then it must enter all the $k-1$ cycles adjacent to $P^i$ through $P_v$. Again, we cannot yet exclude a scenario that this would happen for multiple vertices $v \in V^i$.

▶ **Lemma 11.** *For any Hamiltonian cycle $H$ in $G'$ and $i \in [k]$, there exists a subset $U \subseteq V^i$ for which we have $E(H) \cap E(P^i, \bigcup_{j\neq i} C^{i\to j}) = \bigcup_{v\in U} E(P_v, \bigcup_{j\neq i} C^{i\to j})$.*

**Proof.** Targeting a contradiction, suppose there exists a vertex $v \in V^i$ such that at least one edge of $E(P_v, \bigcup_{j\neq i} C^{i\to j})$ is in $E(H)$ and at least one edge of $E(P_v, \bigcup_{j\neq i} C^{i\to j})$ is not in $E(H)$. From Lemma 10 we know that $E(H) \cap E(P_v, \bigcup_{j\neq i} C^{i\to j})$ is a union of pairs of the form $\{e_v^{i\to j,\mathsf{out}}, e_v^{i\to j,\mathsf{in}}\}$ for some $j \in [k] \setminus \{i\}$. Hence there indices $p, j \in [k] \setminus \{i\}$ so that $(v^{p,\mathsf{in}}, v^{j,\mathsf{out}}) \in E(P^i)$, one of the sets $E(B_v, C^{i\to p})$, $E(B_v, C^{i\to j})$ has empty intersection with $E(H)$ and the other one is contained in $E(H)$. We can assume w.l.o.g. that $j = p+1$ and the first of these sets is empty.

We arrive at the following scenario: $p \in [k] \setminus \{i\}$, $e_v^{i\to p,\mathsf{out}}, e_v^{i\to p,\mathsf{in}} \notin E(H)$ and $e_v^{i\to p+1,\mathsf{out}}, e_v^{i\to p+1,\mathsf{in}} \in E(H)$. Since $H$ is a Hamiltonian cycle, it has to visit the vertex $v^{p,\mathsf{in}}$. Thus, it has to traverse an outgoing edge of $v^{p,\mathsf{in}}$. The only outgoing edges of $v^{p,\mathsf{in}}$ are the edges $e_v^{i\to p,\mathsf{in}}$ and $(v^{p,\mathsf{in}}, v^{p+1,\mathsf{out}})$. By the assumption, $e_v^{i\to p,\mathsf{in}} \notin E(H)$ so $(v^{p,\mathsf{in}}, v^{p+1,\mathsf{out}}) \in E(H)$. On the other hand, since $e_v^{i\to p+1,\mathsf{out}} \in E(H)$ is an incoming edge of $v^{p+1,\mathsf{out}}$, we have $(v^{p,\mathsf{in}}, v^{p+1,\mathsf{out}}) \notin E(H)$ as it is also an incoming edge to $v^{p+1,\mathsf{out}}$. This yields a contradiction. ◀

We want to argue now that in fact the set $U$ in Lemma 11 is a singleton, that is, for fixed $i \in [k]$ each cycle $C^{i\to j}$ is being entered exactly once and from the same subpath $P_v$ of $P^i$. To this end, we take advantage of the universal vertices $u^i$.

▶ **Lemma 12.** *For any Hamiltonian cycle $H$ in $G'$ and $i \in [k]$ there exists $v \in V^i$ such that $E(H) \cap E(P^i, \bigcup_{j\neq i} C^{i\to j})$ is $E(P_v, \bigcup_{j\neq i} C^{i\to j})$.*

**Proof.** Let $U$ be the set from Lemma 11. Targeting a contradiction, suppose that there exist two distinct $v, w \in U$. That is, $E(H) \cap E(P^i, \bigcup_{j\neq i} C^{i\to j}) \supseteq E(P_v, \bigcup_{j\neq i} C^{i\to j}) \cup E(P_w, \bigcup_{j\neq i} C^{i\to j})$.

The Hamiltonian cycle $H$ must contain an outgoing edge of $v^{left}$, which is the first vertex on the path $P_v$. The only out-neighbors of $v^{left}$ are $u^i$ and $v^{p,\mathsf{out}}$ where $p = 1$ when $i \neq 1$ or $p = 2$ when $i = 1$. Recall that the $e_v^{i\to p,\mathsf{out}}$ is present in $E(B_v^i, \bigcup_{j\neq i} C^{i\to j})$ and thereby in $E(H)$ by the assumption. Also, since the edge $e_v^{i\to p,\mathsf{out}}$ is an incoming edge of $v^{p,\mathsf{out}}$ , the edge $(v^{left}, v^{p,\mathsf{out}})$ cannot be in $E(H)$. Thus the outgoing edge of $v^{left}$ in $H$ is $(v^{left}, u^i)$.

Now consider the vertex $w^{left}$, which is the first one of the path $P_w$. By the same argument as above, we have $(w^{left}, u^i) \in E(H)$. This implies that $u^i$ has two in-neighbors in $H$, a contradiction.

It is also impossible that $U = \emptyset$ because then $E(H) \cap \partial(C^{i\to j}) = \emptyset$ for each $j \neq i$ implying that $H$ is disconnected. Consequently, $U$ contains exactly one element. ◀

We can summarize the arguments given so far as follows: for a Hamiltonian cycle $H$ in $G'$ and $i \in [k]$, there exists $v_i \in V^i$, such that, for all $j \neq i$, it holds that $E(H) \cap \partial(C^{i\to j}) = E(P_{v_i}, C^{i\to j}) = \{e_{v_i}^{i\to j,\mathsf{out}}, e_{v_i}^{i\to j,\mathsf{in}}\}$. We make note of a simple implication of this fact.

▶ **Lemma 13.** *Let $H$ be a Hamiltonian cycle in $G'$ and $i, j \in [k]$. Let $v \in V^i$ satisfy $E(H) \cap \partial(C^{i\to j}) = \{e_v^{i\to j,\mathsf{out}}, e_v^{i\to j,\mathsf{in}}\}$. Then $e = (v^{j,\mathsf{out}}, v^{j,\mathsf{in}})$ does not belong to $E(H)$.*

**Proof.** Since $E(H) \cap \partial(C^{i \to j})$ comprises exactly two edges, we infer that $H$ contains the path $Q = v^{j,\text{in}} \to c_v^{i \to j,\text{in}} \to \ldots \to c_v^{i \to j,\text{out}} \to v^{j,\text{out}}$ with all internal vertices from $V(C^{i \to j})$. If $H$ traversed the edge $e = (v^{j,\text{out}}, v^{j,\text{in}})$, then it would contain the cycle $C$ formed by $Q$ and $e$. This would imply $H = C$, which contradicts that $H$ is Hamiltonian. ◀

We are going to show that $H$ can include only one edge that goes from $V(P^i)$ to $V(P^j)$; it will follow that this edge must be $(v_i^{j,\text{out}}, v_j^{i,\text{out}})$.

▶ **Lemma 14.** *Let $H$ be a Hamiltonian cycle in $G'$. For each pair $i, j \in [k]$ with $i < j$ we have $|E(H) \cap E(P^i, P^j)| \leq 1$.*

**Proof.** Suppose that $|E(H) \cap E(P^i, P^j)| \geq 2$. Then there exist $u, w \in V^j$ and $e_u \in \partial^{\text{in}}(u^{i,\text{in}}) \cap \partial^{\text{out}}(P^i)$, $e_w \in \partial^{\text{in}}(w^{i,\text{in}}) \cap \partial^{\text{out}}(P^i)$ such that $e_u, e_w \in E(H)$. This implies that the edges $(u^{i,\text{out}}, u^{i,\text{in}})$, $(w^{i,\text{out}}, w^{i,\text{in}}) \in E(P^j)$ cannot be used by $H$. Since $i < j$, the vertex $u^{i,\text{out}}$ has only one out-neighbor different than $u^{i,\text{in}}$: the terminal $\widehat{t}^{i \to j}$. Hence $(u^{i,\text{out}}, \widehat{t}^{i \to j}) \in E(H)$. But the same argument applies to $w^{i,\text{out}}$. As a consequence, two incoming edges of $\widehat{t}^{i \to j}$ are being used by $H$ and so we arrive at a contradiction. ◀

Finally, we prove the second implication in the correctness proof of the reduction.

▶ **Lemma 15.** *If $(G', X)$ is a Yes-instance of* Hamiltonian Cycle By DFVS *then $(G, (V^1, V^2, \ldots, V^k))$ is a Yes-instance of* Multicolored Clique.

**Proof.** Let $H$ be a Hamiltonian cycle in $G'$ and $v_i \in V^i$ be the vertex given by Lemma 12 for $i \in [k]$. Fix a pair of indices $i < j$. By Lemma 13 we know that the edge $(v_i^{j,\text{out}}, v_i^{j,\text{in}})$ does not belong to $E(H)$. The remaining outgoing edges of $v_i^{j,\text{out}}$ belong to $E(P^i, P^j)$ and $H$ must utilize one of them. By the same argument, $H$ must traverse one of the incoming edges of $v_j^{i,\text{in}}$ that belongs to $E(P^i, P^j)$. Due to Lemma 14, the Hamiltonian cycle $H$ can use at most one edge from $E(P^i, P^j)$. Consequently, we obtain that $(v_i^{j,\text{out}}, v_j^{i,\text{in}}) \in E(H)$. In particular, this means that $(v_i^{j,\text{out}}, v_j^{i,\text{in}})$ is present in $E(G')$ and, by the construction of $G'$, implies that $v_i v_j \in E(G)$. Therefore, $\{v_1, v_2, \ldots, v_k\}$ forms a clique in $G$. ◀

Lemmas 9 and 15 constitute that the instances $(G, (V^1, V^2, \ldots, V^k))$ and $(G', X)$ are equivalent while Lemma 8 ensures that $|X| = \mathcal{O}(k^2)$. We have thus obtained a parameterized reduction from Multicolored Clique to Hamiltonian Cycle By DFVS, proving Theorem 1.

Due to space constraints, the W[1]-hardness proofs for Longest Cycle Above Girth and the path variants of both problems are provided in the full version.

## 4 XP Algorithm for Longest Path Above Girth

Johnson et al. [58] proved that Hamiltonian Path is in XP parameterized by directed treewidth. This result was later extended by de Oliveira Oliveira [30] to capture a wider range of problems expressible in Monadic Second Order (MSO) logic with counting constraints. As a special case of this theorem, we have the following.

▶ **Theorem 16** ([30]). Longest Path *on directed graphs is in* XP *when parameterized by directed treewidth.*

We will use the Directed Grid Theorem (Theorem 7), which either returns a cylindrical wall of order $\mathcal{O}(k)$ in $G$ or concludes that the directed treewidth of $G$ is bounded in terms of $k$. In the former case, we prove that a path of length $g \cdot k$ always exists in $G$. In the latter case, we use Theorem 16 to solve the problem optimally by an XP algorithm.

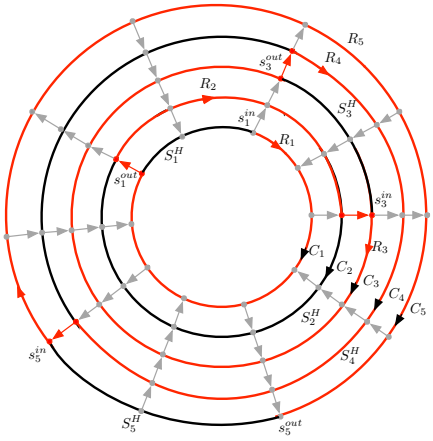



**Figure 5** The path $R$ (colored in red) in a subdivision of $W_5$, described in Lemma 20.

Throughout this section we abbreviate $W = W_{2k+1}$ (the cylindrical wall of order $2k+1$). In order to establish correctness of the algorithm, we need to argue that any subdivision $H$ of $W$ admits a path $k$ times longer than the girth of $H$.

We denote the $2k+1$ vertex disjoint cycles in $W$ by $C_1^W, C_2^W, \ldots C_{2k+1}^W$ counting from the innermost one. For a subdivision $H$ of $W$ we denote the counterpart of $C_i^W$ in $H$ as $C_i^H$. We refer to the girth of $H$ as $g$. Note that $g$ lower bounds the length of each cycle $C_i^H$.

▶ **Definition 17.** *A subpath in $C_i^W$ is called a* segment *of $C_i^W$ if its endpoints have out-neighbors in $C_{i+1}^W$ and none of its internal vertices has out-neighbors in $C_{i+1}^W$. A subpath in $C_i^H$ is a* segment *of $C_i^H$ if it is a subdivision of a segment in $C_i^W$.*

We make note of a few properties of segments.

▶ **Observation 18.** *For every $i \in [2k+1]$ the following hold.*

1. *Each cycle $C_i^W$ is a cyclic concatenation of $k+1$ segments of length 4.*
2. *Each cycle $C_i^H$ is a cyclic concatenation of $k+1$ segments.*
3. *If $i > 1$ then each segment of $C_i^W$ has a unique internal vertex with an in-neighbor in $C_{i-1}^W$.*

We show that in every cycle $C_i^H$ we can choose a segment $S$ so that the path obtained by traversing all vertices in $C_i^H$ that are not internal vertices of $S$ is almost as long as $C_i^H$.

▶ **Lemma 19.** *For a segment $S$ of $C_i^H$, let $\widehat{S}$ be the subpath of $C_i^H$ that starts at the end of $S$ and ends at the start of $S$. Then there exists a segment $S_i^H$ of $C_i^H$ such that $\widehat{S}_i^H$ has length at least $g - \frac{g}{k+1}$.*

**Proof.** By Observation 18 we know that $C_i^H$ can be partitioned into $k+1$ segments. By a counting argument, there exist a segment $S_i^H$ of size at most $\frac{|E(C_i^H)|}{k+1}$. Thus, for the path $\widehat{S}_i^H$ complementing $S_i^H$, we have $|E(\widehat{S}_i^H)| \geq |E(C_i^H)| \cdot (1 - \frac{1}{k+1}) \geq g \cdot (1 - \frac{1}{k+1})$. ◀

Let $s_i^{out}$ denote the first vertex on $S_i^H$. If $i > 1$, we define $s_i^{in}$ to be the unique internal vertex of $S_i^H$ corresponding to a vertex in $C_i^W$ with an in-neighbor in $C_{i-1}^W$. If $i = 1$, we define $s_i^{in}$ to be the last vertex on $S_i^H$.

▶ **Lemma 20.** *Let $H$ be a subdivision of $W_{2k+1}$ and $g$ be the girth of $H$. Then there exists a path of length at least $g \cdot k$ in $H$.*

**Proof.** For $i \in [2k+1]$ we define path $R_i$ in $C_i$ as follows. If $i$ is odd, we set $R_i$ to be the subpath of $C_i$ from $s_i^{in}$ to $s_i^{\circ out}$. If $i$ is even, the path $R_i$ begins at the unique vertex in $C_i^H$ that can be reached via a subdivided edge from $s_{i-1}^{\circ out}$ and $R_i$ ends at the unique vertex in $C_i^H$ from which $s_{i+1}^{in}$ is reachable by via a subdivided edge. Note that the first and last indices in $[2k+1]$ are odd, so the paths $R_i$ are well-defined. We can now concatenate $R_1, R_2, \ldots, R_{2k+1}$ using the subdivided edges between the cycles to obtain a path $R$ in $H$; see Figure 5.

We argue that $R$ is sufficiently long. There are $k+1$ odd indices in $[2k+1]$, and, for each of them, $R_i$ contains the path $\widehat{S}_i^H$ (from Lemma 19) of length at least $g - \frac{g}{k+1}$. Since these form vertex disjoint subpaths of $R$, we conclude that $E(R) \geq (g - \frac{g}{k+1})(k+1) = g(k+1) - g = g \cdot k$. ◀

We are ready to summarize the entire algorithm.

▶ **Theorem 3.** Longest Path Above Girth *is in* XP.

**Proof.** We first execute the algorithm from Theorem 7, which returns either a directed tree decomposition of width $f(k)$, for some computable function $f$, or a subgraph $H$ of $G$ that is a subdivision of $W_{2k+1}$. Note that the girth of $H$ is at least the girth of $G$. In the first case, we apply the algorithm from Theorem 16 to find the longest path in $G$ in polynomial time for fixed $k$. In the latter case, Lemma 20 asserts that $H$ contains a path of length at least $k$ times the girth of $H$, so we can report that $(G, k)$ a Yes-instance. ◀

## References


1 Isolde Adler. Directed tree-width examples. *Journal of Combinatorial Theory, Series B*, 97(5):718–725, 2007.

2 Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the hamiltonian cycle problem for bipartite graphs. *Journal of Information processing*, 3(2):73–76, 1980.

3 Noga Alon and Shai Gutner. Balanced hashing, color coding and approximate counting. In *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, pages 1–16, 2009.

4 Noga Alon and Shai Gutner. Balanced families of perfect hash functions and their applications. *ACM Trans. Algorithms*, 6(3):54:1–54:12, 2010.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

6 Vikraman Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In *Algorithms and Computation: 13th International Symposium, ISAAC 2002 Vancouver, BC, Canada, November 21–23, 2002 Proceedings 13*, pages 453–464. Springer, 2002.

7 Benjamin Bergougnoux, Mamadou Moustapha Kanté, and O-joung Kwon. An optimal xp algorithm for hamiltonian cycle on graphs of bounded clique-width. *Algorithmica*, 82(6):1654–1674, 2020.

8 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding detours is fixed-parameter tractable. *SIAM J. Discret. Math.*, 33(4):2326–2345, 2019.

9 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

10 Andreas Björklund. Exploiting sparsity for bipartite hamiltonicity. In *29th International Symposium on Algorithms and Computation, ISAAC 2018, December 16-19, 2018, Jiaoxi, Yilan, Taiwan*, pages 3:1–3:11, 2018.

11 Andreas Björklund. An asymptotically fast polynomial space algorithm for hamiltonicity detection in sparse directed graphs. In *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, pages 15:1–15:12, 2021.

12 Andreas Björklund and Thore Husfeldt. The parity of directed hamiltonian cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735, 2013.

13 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. In *Algorithms-ESA 2009: 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings 17*, pages 578–586. Springer, 2009.

14 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Narrow sieves for parameterized paths and packings. *J. Comput. Syst. Sci.*, 87:119–139, 2017.

15 Andreas Björklund, Petteri Kaski, and Ioannis Koutis. Directed hamiltonicity and out-branchings via generalized laplacians. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 91:1–91:14, 2017.

16 Andreas Björklund and Ryan Williams. Computing permanents and counting hamiltonian cycles by listing dissimilar vectors. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 25:1–25:14, 2019.

17 Hans L. Bodlaender. On linear time minor tests with depth-first search. *Journal of Algorithms*, 14(1):1–23, 1993.

18 Cornelius Brand, Holger Dell, and Thore Husfeldt. Extensor-coding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 151–164, 2018.

19 Cornelius Brand and Kevin Pratt. Parameterized applications of symbolic differentiation of (totally) multilinear polynomials. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, pages 38:1–38:19, 2021.

20 Michael Buro. Simple amazons endgames and their connection to hamilton circuits in cubic subgrid graphs. In *Computers and Games: Second International Conference, CG 2000 Hamamatsu, Japan, October 26–28, 2000 Revised Papers 2*, pages 250–261. Springer, 2001.

21 Victor Campos, Raul Lopes, Ana Karolinna Maia, and Ignasi Sau. Adapting the directed grid theorem into an fpt algorithm. *Electronic Notes in Theoretical Computer Science*, 346:229–240, 2019.

22 Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009.

23 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008.

24 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. doi:10.1017/CBO9780511977619.

25 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 210–223, 2017.

26 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 130–139. IEEE, 2014.

27 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

28 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *Journal of the ACM (JACM)*, 65(3):1–46, 2018.

29 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan MM Van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Transactions on Algorithms (TALG)*, 18(2):1–31, 2022.

**30** Mateus de Oliveira Oliveira. An algorithmic metatheorem for directed treewidth. *Discrete Applied Mathematics*, 204:49–76, 2016. doi:10.1016/j.dam.2015.10.020.

**31** Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

**32** Frederic Dorn, Fedor V Fomin, and Dimitrios M Thilikos. Catalan structures and dynamic programming in h-minor-free graphs. *Journal of Computer and System Sciences*, 78(5):1606–1622, 2012.

**33** Martin Doucha and Jan Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *MFCS*, volume 2012, pages 348–359. Springer, 2012.

**34** Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

**35** Fedor V. Fomin, Petr A. Golovach, William Lochet, Danil Sagunov, Kirill Simonov, and Saket Saurabh. Detours in directed graphs. In *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, pages 29:1–29:16, 2022.

**36** Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Multiplicative parameterization above a guarantee. *ACM Trans. Comput. Theory*, 13(3):18:1–18:16, 2021.

**37** Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019.

**38** Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Algorithmic extensions of dirac's theorem. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 406–416, 2022.

**39** Fedor V. Fomin, Petr A. Golovach, Danil Sagunov, and Kirill Simonov. Longest cycle above erdős-gallai bound. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, pages 55:1–55:15, 2022.

**40** Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Long directed $(s, t)$-path: FPT algorithm. *Inf. Process. Lett.*, 140:8–12, 2018.

**41** Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Decomposition of map graphs with applications. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 60:1–60:15, 2019.

**42** Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discret. Comput. Geom.*, 62(4):879–911, 2019.

**43** Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

**44** Harold N Gabow and Shuxin Nie. Finding a long directed cycle. *ACM Transactions on Algorithms (TALG)*, 4(1):1–21, 2008.

**45** Robert Ganian. Improving vertex cover as a graph parameter. *Discrete Mathematics & Theoretical Computer Science*, 17, 2015.

**46** Michael R Garey, David S Johnson, and Larry Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing (STOC)*, pages 47–63, 1974.

**47** Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-joung Kwon. The directed flat wall theorem. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 239–258, 2020.

**48** Archontia C. Giannopoulou, Ken-ichi Kawarabayashi, Stephan Kreutzer, and O-joung Kwon. Directed tangle tree-decompositions and applications. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 377–405, 2022.

**49** Petr A. Golovach, R. Krithika, Abhishek Sahu, Saket Saurabh, and Meirav Zehavi. Graph hamiltonicity parameterized by proper interval deletion set. In *LATIN 2020: Theoretical Informatics - 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, pages 104–115, 2020.

**50** Ronald J Gould. Advances on the hamiltonian problem–a survey. *Graphs and Combinatorics*, 19(1):7–52, 2003.

**51** Gregory Gutin and Matthias Mnich. A survey on graph problems parameterized above and below guaranteed values. *arXiv preprint arXiv:2207.12278*, 2022.

**52** Meike Hatzel, Ken-ichi Kawarabayashi, and Stephan Kreutzer. Polynomial planar directed grid theorem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1465–1484, 2019.

**53** Meike Hatzel, Konrad Majewski, Michal Pilipczuk, and Marek Sokolowski. Simpler and faster algorithms for detours in planar digraphs. In *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 156–165, 2023.

**54** Falk Hüffner, Sebastian Wernicke, and Thomas Zichner. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica*, 52(2):114–132, 2008.

**55** Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008.

**56** Ashwin Jacob, Michal Wlodarczyk, and Meirav Zehavi. Long directed detours: Reduction to 2-disjoint paths. *CoRR*, abs/2301.06105, 2023.

**57** Bart M. P. Jansen, László Kozma, and Jesper Nederlof. Hamiltonicity below dirac's condition. In *Graph-Theoretic Concepts in Computer Science - 45th International Workshop, WG 2019, Vall de Núria, Spain, June 19-21, 2019, Revised Papers*, pages 27–39, 2019.

**58** Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

**59** Georgia Kaouri, Michael Lampis, and Valia Mitsou. New directions in directed treewidth. *Parameterized Complexity News: Newsletter of the Parameterized Complexity Community*, September:4–5, 2009.

**60** Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103, 1972.

**61** Ken-ichi Kawarabayashi and Stephan Kreutzer. The directed grid theorem. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 655–664, 2015.

**62** Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 575–586, 2008.

**63** Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. In *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings*, pages 220–231, 2008.

**64** Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discret. Optim.*, 8(1):129–138, 2011.

**65** Daniel Lokshtanov, Andreas Björklund, Saket Saurabh, and Meirav Zehavi. Approximate counting of $k$-paths: Simpler, deterministic, and in polynomial space. *ACM Trans. Algorithms*, 17(3):26:1–26:44, 2021.

**66** Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Planar k-path in subexponential time and polynomial space. In *WG*, pages 262–270. Springer, 2011.

**67** Burkhard Monien. How to find long paths efficiently. In *North-Holland Mathematics Studies*, volume 109, pages 239–254. Elsevier, 1985.

**68** Jesper Nederlof. Detecting and counting small patterns in planar graphs in subexponential parameterized time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1293–1306, 2020.

**69** Jan Obdrzálek. Dag-width: connectivity measure for directed graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 814–821, 2006.

**70** Bruce A. Reed. Introducing directed tree width. *Electron. Notes Discret. Math.*, 3:222–229, 1999.

**71** Neil Robertson and Paul D Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

**72** Hadas Shachnai and Meirav Zehavi. Representative families: A unified tradeoff-based approach. *J. Comput. Syst. Sci.*, 82(3):488–502, 2016.

**73** Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM J. Discret. Math.*, 24(1):146–157, 2010. doi:10.1137/070697781.

**74** Dekel Tsur. Faster deterministic parameterized algorithm for $k$-path. *Theor. Comput. Sci.*, 790:96–104, 2019.

**75** Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proceedings of the forty-first annual ACM symposium on Theory of computing (STOC)*, pages 455–464, 2009.

**76** Sebastian Wiederrecht. A note on directed treewidth. *CoRR*, abs/1910.01826, 2019. arXiv:1910.01826.

**77** Ryan Williams. Finding paths of length k in $\mathcal{O}^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

**78** Meirav Zehavi. Mixing color coding-related techniques. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 1037–1049, 2015.

**79** Meirav Zehavi. A randomized algorithm for long directed cycle. *Inf. Process. Lett.*, 116(6):419–422, 2016.

**80** Meirav Zehavi, Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Eth-tight algorithms for long path and cycle on unit disk graphs. *J. Comput. Geom.*, 12(2):126–148, 2021.

# 5-Approximation for $\mathcal{H}$-Treewidth Essentially as Fast as $\mathcal{H}$-Deletion Parameterized by Solution Size

**Bart M. P. Jansen**
Eindhoven University of Technology, The Netherlands

**Jari J. H. de Kroon**
Eindhoven University of Technology, The Netherlands

**Michał Włodarczyk**
University of Warsaw, Poland

## Abstract

The notion of $\mathcal{H}$-treewidth, where $\mathcal{H}$ is a hereditary graph class, was recently introduced as a generalization of the treewidth of an undirected graph. Roughly speaking, a graph of $\mathcal{H}$-treewidth at most $k$ can be decomposed into (arbitrarily large) $\mathcal{H}$-subgraphs which interact only through vertex sets of size $\mathcal{O}(k)$ which can be organized in a tree-like fashion. $\mathcal{H}$-treewidth can be used as a hybrid parameterization to develop fixed-parameter tractable algorithms for $\mathcal{H}$-DELETION problems, which ask to find a minimum vertex set whose removal from a given graph $G$ turns it into a member of $\mathcal{H}$. The bottleneck in the current parameterized algorithms lies in the computation of suitable tree $\mathcal{H}$-decompositions.

We present FPT-approximation algorithms to compute tree $\mathcal{H}$-decompositions for hereditary and union-closed graph classes $\mathcal{H}$. Given a graph of $\mathcal{H}$-treewidth $k$, we can compute a 5-approximate tree $\mathcal{H}$-decomposition in time $f(\mathcal{O}(k)) \cdot n^{\mathcal{O}(1)}$ whenever $\mathcal{H}$-DELETION parameterized by solution size can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$ for some function $f(k) \geq 2^k$. The current-best algorithms either achieve an approximation factor of $k^{\mathcal{O}(1)}$ or construct optimal decompositions while suffering from non-uniformity with unknown parameter dependence. Using these decompositions, we obtain algorithms solving ODD CYCLE TRANSVERSAL in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ parameterized by bipartite-treewidth and VERTEX PLANARIZATION in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ parameterized by planar-treewidth, showing that these can be as fast as the solution-size parameterizations and giving the first ETH-tight algorithms for parameterizations by hybrid width measures.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** fixed-parameter tractability, treewidth, graph decompositions

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.66

**Related Version** *Full Version*: https://arxiv.org/abs/2306.17065 [31]

**Funding** This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).

## 1 Introduction

**Background and motivation.** Treewidth (see [7, 19] [16, §7]) is a width measure for graphs that is ubiquitous in algorithmic graph theory. It features prominently in the Graph Minors series [47] and frequently pops up unexpectedly [38] in the parameterized complexity [16, 20, 23] analysis of NP-hard graph problems on undirected graphs. The notion of treewidth captures how tree-like a graph is in a certain sense; it is defined as the width of an optimal tree decomposition for the graph. Unfortunately, computing an optimal tree decomposition is NP-hard [3]. As many of the algorithmic applications of treewidth

require a tree decomposition to be able to work, there has been long record of algorithms computing optimal [3, 6, 9, 43] or near-optimal [4, 8, 35] tree decompositions with no end in sight [36], as well as a long series of experimental work on heuristically computing good tree decompositions [10, 11, 17, 18]. In this paper, we present a new fixed-parameter tractable approximation algorithm for the notion of $\mathcal{H}$-treewidth, a generalization of treewidth which has recently attracted significant attention [1, 21, 29, 30]. Before describing our contributions for $\mathcal{H}$-treewidth, we summarize the most important background to motivate the problem.

The popularity of treewidth as a graph parameter can be attributed to the fact that it has very good algorithmic properties (by Courcelle's theorem, any problem that can be formulated in Counting Monadic Second-Order ($\mathrm{CMSO}_2$) logic can be solved in linear time on graphs of bounded treewidth [15]), while also having a very elegant mathematical structure theory. Unfortunately, simple substructures like grids or cliques in a graph can already make its treewidth large. This means that for many input graphs of interest, the treewidth is too large for an approach based on treewidth to be efficient: the running times of many treewidth-based algorithms are of the form $f(k) \cdot n^{\mathcal{O}(1)}$, where $f$ is an exponential function in the treewidth $k$ and $n$ is the total number of vertices of the graph.

Several approaches have been taken to cope with the fact that treewidth is large on graphs with large cliques or large induced grid subgraphs. One approach lies in generalized width measures like cliquewidth or rankwidth [41], by essentially replacing the use of separations of small order (which are encoded in tree decompositions), by separations of large order but in which the interactions between the two sides is well-structured. Unfortunately this generality comes at a price in terms of algorithmic applications [24, 25, 26].

This has recently led Eiben, Ganian, Hamm, and Kwon [21] to enrich the notion of treewidth in a different way. Consider a hereditary class $\mathcal{H}$ of graphs, such as bipartite graphs. The notion of $\mathcal{H}$-treewidth aims to capture how well a graph $G$ can be decomposed into subgraphs belonging to $\mathcal{H}$ which only interact with the rest of the graph via small vertex sets which are organized in a tree-like manner. While we defer formal definitions of $\mathcal{H}$-treewidth to Section 2, an intuitive way to think of the concept is the following: a graph $G$ has $\mathcal{H}$-treewidth at most $k$ if and only if it can be obtained from a graph $G_0$ with a tree decomposition of width at most $k$ by the following process: repeatedly insert a subgraph $H_i$ belonging to graph class $\mathcal{H}$, such that the neighbors of $H_i$ in the rest of the graph are all contained in a single bag of the tree decomposition of $G_0$. The $\mathcal{H}$-subgraphs $H_i$ inserted during this process are called *base components* and their neighborhoods have size at most $k+1$. When $\mathcal{H}$ is a graph class of unbounded treewidth, like bipartite graphs, the $\mathcal{H}$-treewidth of a graph can be arbitrarily much smaller than its treewidth. This prompted an investigation of the algorithmic applications of $\mathcal{H}$-treewidth.

In recent works [1, 21, 30], the notion of $\mathcal{H}$-treewidth was used to develop new algorithms to solve vertex-deletion problems. Many classic NP-hard problems in algorithmic graph theory can be phrased in the framework of $\mathcal{H}$-deletion: find a minimum vertex-subset $S$ of the input graph $G$ such that $G - S$ belongs to a prescribed graph class $\mathcal{H}$. Examples include Vertex Cover (where $\mathcal{H}$ is the class of edgeless graphs), Odd Cycle Transversal (bipartite graphs), and Vertex Planarization (planar graphs). All these problems are known to be fixed-parameter tractable [14, 33, 34, 39, 44] when parameterized by the size of a desired solution: there are algorithms that, given an $n$-vertex graph $G$ and integer $k$, run in time $f(k) \cdot n^{\mathcal{O}(1)}$ and output a vertex set $S \subseteq V(G)$ of size at most $k$ for which $G - S \in \mathcal{H}$, if such a set exists. These algorithms show that large instances whose optimal solutions are small, can still be solved efficiently. Alternatively, since the mentioned graph classes $\mathcal{H}$ can be defined in $\mathrm{CMSO}_2$, these vertex-deletion problems can be solved in time $f(w) \cdot n$

parameterized by the treewidth $w$ of the input graph via Courcelle's theorem, which shows that instances of small treewidth (but whose optimal solutions may be large) can be solved efficiently.

The notion of $\mathcal{H}$-treewidth (abbreviated as $\mathbf{tw}_{\mathcal{H}}$ from now on) can be used to combine the best of both worlds. It is not difficult to show that if a graph $G$ has a vertex set $S$ of size $k$ for which $G - S \in \mathcal{H}$ (we call such a set an $\mathcal{H}$-deletion set), then the $\mathcal{H}$-treewidth of $G$ is at most $k$: simply take a trivial tree decomposition consisting of a single bag of size $k$ for the graph $G_0 := G[S]$, so that afterwards the graph $G$ can be obtained from $G_0$ by inserting the graph $H = G - S$, which belongs to $\mathcal{H}$ and has all its neighbors in a single bag. Since the $\mathcal{H}$-treewidth of $G$ is also never larger than its standard treewidth, the *hybrid* (cf. [2]) parameterization by $\mathcal{H}$-treewidth dominates both the parameterizations by the solution size and the treewidth of the graph. This raises the question whether existing fixed-parameter tractability results for parameterizations of $\mathcal{H}$-DELETION by treewidth or solution size, can be extended to $\mathbf{tw}_{\mathcal{H}}$.

It was recently shown [1] that when it comes to *non-uniform* fixed-parameter tractability characterizations, the answer to this question is positive. If $\mathcal{H}$ satisfies certain mild conditions, which is the case for all graph classes mentioned so far, then for each value of $k$ there exists an algorithm $\mathcal{A}_{\mathcal{H},k}$ that, given a graph $G$ with $\mathbf{tw}_{\mathcal{H}}(G) \leq k$ and target value $t$, decides whether or not $G$ has an $\mathcal{H}$-deletion set of size at most $t$. There is a constant $c_{\mathcal{H}}$ such that each algorithm $\mathcal{A}_{\mathcal{H},k}$ runs in time $\mathcal{O}(n^{c_{\mathcal{H}}})$, so that the overall running time can be bounded by $f(k) \cdot n^{c_{\mathcal{H}}}$; however, no bounds on the function $f$ are given and in general it is unknown how to construct the algorithms whose existence is proven. Another recent paper [29] gave concrete FPT algorithms to solve $\mathcal{H}$-DELETION parameterized by $\mathbf{tw}_{\mathcal{H}}$ for certain cases of $\mathcal{H}$, including the three mentioned ones. For example, it presents an algorithm that solves ODD CYCLE TRANSVERSAL in time $2^{\mathcal{O}(k^3)} \cdot n^{\mathcal{O}(1)}$, parameterized by bipartite-treewidth. The bottleneck in the latter approach lies in the computation of a suitable tree $\mathcal{H}$-decomposition: on a graph of $\mathbf{tw}_{\mathcal{H}}(G) \leq k$, the algorithm runs in $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ time to compute a tree $\mathcal{H}$-decomposition of width $w \in \mathcal{O}(k^3)$, and then optimally solves ODD CYCLE TRANSVERSAL on the decomposition of width $w$ in time $2^{\mathcal{O}(w)} \cdot n^{\mathcal{O}(1)} \leq 2^{\mathcal{O}(k^3)} \cdot n^{\mathcal{O}(1)}$. Note that the parameter dependence of this algorithm is much worse than for the parameterizations by solution size and by treewidth, both of which can be solved in single-exponential time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ [44, 37]. To improve the running times of algorithms for $\mathcal{H}$-DELETION based on hybrid parameterizations, improved algorithms are therefore required to compute approximate tree $\mathcal{H}$-decompositions. These form the subject of our work.

**Our contribution: $\mathcal{H}$-treewidth.** We develop generic FPT algorithms to approximate $\mathcal{H}$-treewidth, for graph classes $\mathcal{H}$ which are hereditary and closed under taking the disjoint union of graphs. To approximate $\mathcal{H}$-treewidth, all our algorithm needs is access to an oracle for solving $\mathcal{H}$-DELETION parameterized by solution size. The values of the solution size for which the oracle is invoked, will be at most twice as large as the $\mathcal{H}$-treewidth of the graph we are decomposing. Hence existing algorithms for solution-size parameterizations of $\mathcal{H}$-DELETION can be used as a black box to form the oracle. Aside from the oracle calls, our algorithm only takes $8^k \cdot kn(n+m)$ time on an $n$-vertex graph with $m$ edges. So whenever the solution-size parameterization can be solved in single-exponential time, an approximate tree $\mathcal{H}$-decomposition can be found in single-exponential time. The approximation factor of the algorithm is 5, which is a significant improvement over earlier poly(opt) approximations running in superexponential time. The formal statement of our main result is the following.

▶ **Theorem 1.** *Let $\mathcal{H}$ be a hereditary and union-closed class of graphs. There is an algorithm that, using oracle-access to an algorithm $\mathcal{A}$ for $\mathcal{H}$-DELETION, takes as input an $n$-vertex $m$-edge graph $G$, integer $k$, and either computes a tree $\mathcal{H}$-decomposition of $G$ of width at*

*most $5k+5$ consisting of $\mathcal{O}(n)$ nodes, or correctly concludes that $\boldsymbol{tw}_{\mathcal{H}}(G) > k$. The algorithm runs in time $\mathcal{O}(8^k \cdot kn(n+m))$, polynomial space, and makes $\mathcal{O}(8^k n)$ calls to $\mathcal{A}$ on induced subgraphs of $G$ and parameter $2k+2$.*

Theorem 1 yields the first constant-factor approximation algorithms for $\mathbf{tw}_{\mathcal{H}}$ that run in single-exponential time. For example, for $\mathcal{H}$ the class of bipartite graphs the running time becomes $\mathcal{O}(72^k \cdot n^2(n+m))$, and for interval graphs we obtain $\mathcal{O}(8^{3k} \cdot n(n+m))$ (the full version [31] gives results for more classes $\mathcal{H}$). Combining these approximate decompositions with existing algorithms that solve $\mathcal{H}$-DELETION on a given tree $\mathcal{H}$-decomposition, we obtain ETH-tight algorithms as a consequence. ODD CYCLE TRANSVERSAL can be solved in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$, and VERTEX PLANARIZATION can be solved in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ when parameterized by $\mathbf{tw}_{\mathcal{H}}$ for $\mathcal{H}$ the class of bipartite and planar graphs, respectively, without having to supply a decomposition in the input. For VERTEX PLANARIZATION, the previous-best bound [32] was $2^{\mathcal{O}(k^5 \log k)} \cdot n^{\mathcal{O}(1)}$. Note that for the planarization problem, a parameter dependence of $2^{o(k \log k)}$ is impossible assuming the Exponential Time Hypothesis; this already holds for the larger parameterization by treewidth [42]. For ODD CYCLE TRANSVERSAL, an algorithm running in time $2^{o(n)}$ would violate the Exponential Time Hypothesis, which follows by a simple reduction from VERTEX COVER for which such a lower bound is known [16, Theorem 14.6]. This implies that the solution size parameterization cannot be solved in subexponential time.

Compared to existing algorithms to approximate treewidth, the main obstacle we have to overcome in Theorem 1 is identifying the base components of an approximate decomposition in a suitable way. The earlier FPT $\mathsf{poly}(\mathsf{opt})$-approximation for $\mathbf{tw}_{\mathcal{H}}$ effectively reduced the input graph $G$ to a graph $G'$ by repeatedly extracting large $\mathcal{H}$-subgraphs with small neighborhoods, in such a way that the treewidth of $G'$ can be bounded in terms of $\mathbf{tw}_{\mathcal{H}}(G)$, while a tree decomposition of $G'$ can be lifted into an approximate tree $\mathcal{H}$-decomposition of $G$. Several steps in this process led to losses in the approximation factor. To obtain our 5-approximation, we avoid the translation between $G$ and $G'$, and work directly on decomposing the input graph $G$.

Our recursive decomposition algorithm works similarly as the Robertson-Seymour 4-approximation algorithm for treewidth [45] (cf. [16, §7.6]). When given a graph $G$ and integer $k$ with $\mathbf{tw}_{\mathcal{H}}(G) \leq k$, the algorithm maintains a vertex set $S$ of size $3k+4$ which forms the boundary between the part of the graph that has already been decomposed and the part that still needs to be processed. If $S$ has a $\frac{2}{3}$-balanced separator $R$ of size $k+1$, we can proceed in the usual way: we split the graph based on $R$, recursively decompose the resulting parts, and combine these decompositions by adding a bag containing $R \cup S$ as the root. If $S$ does not have a balanced separator of size $k+1$, then we show (modulo some technical details) that for any optimal tree $\mathcal{H}$-decomposition, there is a subset $S' \subseteq S$ of $2k+3$ vertices which belong to a single base component $H_0$. Our main insight is that such a set $S'$ can be used in a win/win approach, by maintaining an $\mathcal{H}$-deletion set $X$ during the decomposition process that initially contains all vertices. To make progress in the recursion, we would like to split off a base component containing $S'$ via a separator $U$ of size at most $2k+2$, while adding $U$ to the boundary of the remainder of the graph to be decomposed. To identify an induced $\mathcal{H}$-subgraph with small neighborhood that can serve as a base component, we compute a minimum $(S', X)$-separator $U$ (we allow $U$ to intersect the sets $S', X$). Any connected component $H$ of $G - U$ that contains a vertex from $S'$ does not contain any vertex of the $\mathcal{H}$-deletion set $X$, so $H$ is an induced subgraph of $G - X$ which implies $H \in \mathcal{H}$ for hereditary $\mathcal{H}$. Hence if there is an $(S', X)$-separator $U$ of size at most $2k+2$, we can use it to split off base components neighboring $U$ that eliminate $2k+3$ vertices from $S$ from the boundary, thereby making room to insert $U$ into the boundary without blowing up its size.

Of course, it may be that all $(S', X)$-separators are larger than $2k + 2$; by Menger's theorem, this happens exactly when there is a family $\mathcal{P}$ of $2k+3$ vertex-disjoint $(S', X)$-paths. Only $k + 1$ paths in $\mathcal{P}$ can *escape* the base component $H_0$ covering $S'$ since its neighborhood has size at most $k + 1$, so that $k + 2$ of them end in a vertex of the deletion set $X$ that lies in $H_0$. The key point is now that this situation implies that $X$ is redundant in a technical sense: if we let $X'$ denote the endpoints of $k + 2$ $(S', X)$-paths starting and ending in $H_0$, we can obtain a smaller $\mathcal{H}$-deletion set by replacing $X'$ by the neighborhood of $H_0$, which has size at most $k + 1$. This replacement is valid as long as $\mathcal{H}$ is hereditary and union-closed. Using an oracle for $\mathcal{H}$-DELETION parameterized by solution size, we can therefore efficiently find a smaller $\mathcal{H}$-deletion set when we know $X'$. While the algorithm does not know $X'$ in general, this type of argument leads to the win/win: either there is a small $(S', X)$-separator which we can use to split off a base component, or there is a large family of vertex-disjoint $(S', X)$-paths which allows the $\mathcal{H}$-deletion set to be improved. As the latter can only happen $|V(G)|$ times, we must eventually identify a base component to split off, allowing the recursion to proceed.

**Our contribution: $\mathcal{H}$-elimination distance.** The $\mathcal{H}$-elimination distance $\mathbf{ed}_{\mathcal{H}}(G)$ of a graph $G$ is a parameter [12, 13] that extends treedepth [40] similarly to how $\mathcal{H}$-treewidth extends treewidth. For hereditary and union-closed classes $\mathcal{H}$, the $\mathcal{H}$-elimination distance of a graph $G$ is the minimum number of rounds needed to turn $G$ into a member of $\mathcal{H}$, when a round consists of removing one vertex from each connected component. Such an elimination process can be represented by a tree structure called *$\mathcal{H}$-elimination forest*. Aside from the fact that computing the $\mathcal{H}$-elimination distance may reveal interesting properties of a graph $G$, a second motivation for studying this parameter is that it can facilitate *polynomial-space* algorithms for solving $\mathcal{H}$-DELETION, while the parameterization by $\mathbf{tw}_{\mathcal{H}}$ (which is never larger) typically gives rise to exponential-space algorithms. At a high level, the state of the art for computing $\mathbf{ed}_{\mathcal{H}}$ is similar as for $\mathbf{tw}_{\mathcal{H}}$: there is an exact non-uniform FPT algorithm with unspecified parameter dependence that works as long as $\mathcal{H}$ satisfies some mild conditions [1], while uniform $\mathsf{poly}(\mathsf{opt})$-approximation algorithms running in time $2^{k^{\mathcal{O}(1)}} \cdot n^{\mathcal{O}(1)}$ are known for several concrete graph classes $\mathcal{H}$ [30].

By leveraging similar ideas as for Theorem 1, we also obtain improved FPT-approximation algorithms for $\mathbf{ed}_{\mathcal{H}}$. The following theorem gives algorithms for two settings: one for an algorithm using polynomial space whenever the algorithm $\mathcal{A}$ for $\mathcal{H}$-DELETION does, which is the case for most of the considered graph classes, and one for an exponential-space algorithm with a better approximation ratio.

▶ **Theorem 2.** *Let $\mathcal{H}$ be a hereditary and union-closed class of graphs. There exists an algorithm that, using oracle-access to an algorithm $\mathcal{A}$ for $\mathcal{H}$-DELETION, takes as input an $n$-vertex graph $G$ and integer $k$, runs in time $n^{\mathcal{O}(1)}$, makes $n^{\mathcal{O}(1)}$ calls to $\mathcal{A}$ on induced subgraphs of $G$ and parameter $2k$, and either concludes that $\mathbf{ed}_{\mathcal{H}}(G) > k$ or outputs an $\mathcal{H}$-elimination forest of depth $\mathcal{O}(k^3 \log^{3/2} k)$.*

*Under the same assumptions, there is an algorithm that runs in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$, makes $n^{\mathcal{O}(1)}$ calls to $\mathcal{A}$ on induced subgraphs of $G$ and parameter $2k$, and either concludes that $\mathbf{ed}_{\mathcal{H}}(G) > k$ or outputs an $\mathcal{H}$-elimination forest of depth $\mathcal{O}(k^2)$.*

In the previous work [30, 32] such a dependence on $k$ was possible only in two cases: when $\mathcal{H}$ is the class of bipartite graphs or when $\mathcal{H}$ is defined by a finite family of forbidden induced subgraphs. In the general case, our result effectively shaves off a single $k$-factor in the depth of the returned decomposition and in the exponent of the running time, compared to the previously known approximations. Theorem 2 entails better approximation algorithms for $\mathcal{H}$-elimination distance for classes of e.g. chordal, interval, planar, bipartite permutation, or distance-hereditary graphs.

**Organization.** The remainder of the paper is organized as follows. We continue by presenting formal preliminaries in Section 2. In Section 3 we treat $\mathcal{H}$-treewidth, developing the theory and subroutines needed to prove Theorem 1. Due to space limitations, the resulting proof of Theorem 1 is deferred to the full version [31], which also provides a list of applications for concrete graph classes. The proof of Theorem 2 can also be found in the full version. We conclude in Section 4.

## 2 Preliminaries

**Graphs and graph classes.** We consider finite, simple, undirected graphs. We denote the vertex and edge sets of a graph $G$ by $V(G)$ and $E(G)$ respectively, with $|V(G)| = n$ and $|E(G)| = m$. For a set of vertices $S \subseteq V(G)$, by $G[S]$ we denote the graph induced by $S$. We use shorthand $G - v$ and $G - S$ for $G[V(G) \setminus \{v\}]$ and $G[V(G) \setminus S]$, respectively. The open neighborhood $N_G(v)$ of $v \in V(G)$ is defined as $\{u \in V(G) \mid uv \in E(G)\}$. The closed neighborhood of $v$ is $N_G[v] = N_G(v) \cup \{v\}$. For $S \subseteq V(G)$, we have $N_G[S] = \bigcup_{v \in S} N_G[v]$ and $N_G(S) = N_G[S] \setminus S$. We define the boundary $\partial_G(S)$ of the vertex set $S$ as $N_G(V(G) \setminus S)$, i.e., those vertices of $S$ which have a neighbor outside $S$.

A class of graphs $\mathcal{H}$ is called *hereditary* if for any $G \in \mathcal{H}$, every induced subgraph of $G$ also belongs to $\mathcal{H}$. Furthermore, $\mathcal{H}$ is *union-closed* if for any $G_1, G_2 \in \mathcal{H}$ the disjoint union of $G_1$ and $G_2$ also belongs to $\mathcal{H}$. For a graph class $\mathcal{H}$ and a graph $G$, a set $X \subseteq V(G)$ is called an $\mathcal{H}$-deletion set in $G$ if $G - X \in \mathcal{H}$. For a graph class $\mathcal{H}$, the parameterized problem $\mathcal{H}$-DELETION takes a graph $G$ and parameter $k$ as input, and either outputs a minimum-size $\mathcal{H}$-deletion set in $G$ or reports that there is no such set of size at most $k$.

**Separators.** For two (not necessarily disjoint) sets $X, Y \subseteq V(G)$ in a graph $G$, a set $P \subseteq V(G)$ is an $(X, Y)$-separator if no connected component of $G - P$ contains a vertex from both $X \setminus P$ and $Y \setminus P$. Such a separator may intersect $X \cup Y$. Equivalently, $P$ is an $(X, Y)$-separator if each $(X, Y)$-path contains a vertex of $P$. The minimum cardinality of such a separator is denoted $\lambda_G(X, Y)$. By Menger's theorem, $\lambda_G(X, Y)$ is equal to the maximum cardinality of a set of pairwise vertex-disjoint $(X, Y)$-paths. A pair $(A, B)$ of subsets of $V(G)$ is a *separation* in $G$ if $A \cup B = V(G)$ and $G$ has no edges between $A \setminus B$ and $B \setminus A$. Its order is defined as $|A \cap B|$.

▶ **Observation 3.** *For two sets $X, Y \subseteq V(G)$, it holds that $\lambda_G(X, Y) \leq k$ if and only if there exists a separation $(A, B)$ in $V(G)$ such that $X \subseteq A$, $Y \subseteq B$, and $|A \cap B| \leq k$.*

The following theorem summarizes how, given vertex sets $X, Y \subseteq V(G)$ and a bound $k$, we can algorithmically find a small-order separation or a large system of vertex-disjoint paths. The statement follows from the analysis of the Ford-Fulkerson algorithm for maximum $(X, Y)$-flow in which each vertex has a capacity of 1. If the algorithm has not terminated within $k$ iterations, then the flow of value $k + 1$ yields $k + 1$ vertex-disjoint paths. If it terminates earlier, a suitable separation can be identified based on reachability in the residual network of the last iteration.

▶ **Theorem 4** (Ford-Fulkerson, see [16, Thm. 8.2] and [49, §9.2])**.** *There is an algorithm that, given an $n$-vertex $m$-edge graph $G$, sets $X, Y \subseteq V(G)$, and integer $k$, runs in time $\mathcal{O}(k(n + m))$ and determines whether $\lambda_G(X, Y) \leq k$. If so, the algorithm also returns a separation $(A, B)$ in $G$ with $X \subseteq A$, $Y \subseteq B$, and $|A \cap B| \leq k$. Otherwise, the algorithm returns a family of $k + 1$ vertex-disjoint $(X, Y)$-paths.*

**$\mathcal{H}$-treewidth.** We continue by giving a formal definition of a tree $\mathcal{H}$-decomposition.

▶ **Definition 5.** *For a graph class $\mathcal{H}$, a tree $\mathcal{H}$-decomposition of graph $G$ is a triple $(T, \chi, L)$ where $L \subseteq V(G)$, $T$ is a rooted tree, and $\chi \colon V(T) \to 2^{V(G)}$, such that:*

1. *For each $v \in V(G)$ the nodes $\{t \mid v \in \chi(t)\}$ form a non-empty connected subtree of $T$.*
2. *For each edge $uv \in E(G)$ there is a node $t \in V(T)$ with $\{u, v\} \subseteq \chi(t)$.*
3. *For each vertex $v \in L$, there is a unique $t \in V(T)$ with $v \in \chi(t)$, and $t$ is a leaf of $T$.*
4. *For each node $t \in V(T)$, the graph $G[\chi(t) \cap L]$ belongs to $\mathcal{H}$.*

*The* width *of a tree $\mathcal{H}$-decomposition is defined as $\max(0, \max_{t \in V(T)} |\chi(t) \setminus L| - 1)$. The $\mathcal{H}$-treewidth of a graph $G$, denoted $\boldsymbol{tw}_{\mathcal{H}}(G)$, is the minimum width of a tree $\mathcal{H}$-decomposition of $G$. The connected components of $G[L]$ are called base components.*

*A pair $(T, \chi)$ is a (standard)* tree decomposition *if $(T, \chi, \emptyset)$ satisfies all conditions of an $\mathcal{H}$-decomposition; the choice of $\mathcal{H}$ is irrelevant.*

For a rooted tree decomposition $(T, \chi)$, $T_t$ denotes the subtree of $T$ rooted at $t \in V(T)$, while $\chi(T_t) = \bigcup_{x \in V(T_t)} \chi(x)$. Similarly as treewidth, $\mathcal{H}$-treewidth is a monotone parameter with respect to taking induced subgraphs.

▶ **Observation 6.** *Let $\mathcal{H}$ be a hereditary class of graphs, $G$ be a graph, and $H$ be an induced subgraph of $G$. Then $\boldsymbol{tw}_{\mathcal{H}}(H) \leq \boldsymbol{tw}_{\mathcal{H}}(G)$.*

## 3 Approximating $\mathcal{H}$-treewidth

We make preparations for the proof of Theorem 1. First, we formalize the concept of a potential base component using the notion of an $(\mathcal{H}, \ell)$-separation and relate it to *redundant* subsets in a solution to $\mathcal{H}$-DELETION. Next, we prove a counterpart of the balanced-separation property for graphs of bounded $\mathcal{H}$-treewidth and explain how it allows us to apply a win/win approach in a single step of the decomposition algorithm.

### 3.1 Redundancy and $(\mathcal{H}, \ell)$-separations

We summon the following concept from the previous work on $\mathcal{H}$-treewidth [30] to capture $\mathcal{H}$-subgraphs with small neighborhoods.

▶ **Definition 7.** *For disjoint $C, S \subseteq V(G)$, the pair $(C, S)$ is called an $(\mathcal{H}, \ell)$-separation in $G$ if (1) $G[C] \in \mathcal{H}$, (2) $|S| \leq \ell$, and (3) $N_G(C) \subseteq S$.*

This notion is tightly connected to the base components of tree $\mathcal{H}$-decompositions. For any tree $\mathcal{H}$-decomposition $(T, \chi, L)$ of width $k$ of a graph $G$, for any node $t \in T$, the graph $G[\chi(t) \cap L]$ belongs to $\mathcal{H}$ so that $C := \chi(t) \cap L$ satisfies Definition 7. The open neighborhood of $\chi(t) \cap L$ is a subset of $S := \chi(t) \setminus L$, which follows from the fact that vertices of $L$ only occur in a single bag, while each edge has both endpoints covered by a single bag. Since $|S| \leq k + 1$ by definition of the width of a tree $\mathcal{H}$-decomposition, this leads to the following observation.

▶ **Observation 8.** *Let $(T, \chi, L)$ be a tree $\mathcal{H}$-decomposition of a graph $G$ of width $k$. For each node $t \in V(T)$, the pair $(\chi(t) \cap L, \chi(t) \setminus L)$ is an $(\mathcal{H}, k+1)$-separation in $G$.*

The following concept will be useful when working with $(\mathcal{H}, \ell)$-separations.

▶ **Definition 9.** *For an $(\mathcal{H}, k)$-separation $(C, S)$ and set $Z \subseteq V(G)$, we say that $(C, S)$ covers $Z$ if $Z \subseteq C$, or weakly covers $Z$ if $Z \subseteq C \cup S$. Set $Z \subseteq V(G)$ is called (weakly) $(\mathcal{H}, \ell)$-separable if there exists an $(\mathcal{H}, \ell)$-separation that (weakly) covers $Z$.*

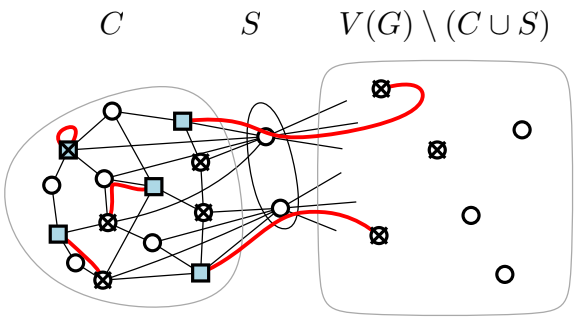



**Figure 1** Illustration for Lemma 12: an $(\mathcal{H}, \ell)$-separation $(C, S)$ in a graph $G$ where $\mathcal{H}$ is the class of triangle-free graphs and $\ell = 2$. Vertices marked with a cross form an $\mathcal{H}$-deletion set $X$ in $G$, while the set $Z$ of size $2\ell + 1 = 5$ marked with blue squares is weakly $(\mathcal{H}, \ell)$-separable. A set of $2\ell+1$ vertex-disjoint $(Z, X)$-paths $\mathcal{P}$ is highlighted, witnessing $\lambda_G(Z, X) = |Z|$. Since $|X \cap C| > \ell$, the set $X$ is not a minimum $\mathcal{H}$-deletion set in $G$: it can be improved by replacing $X \cap C$ with $S$. When $(C, S)$ weakly covering $Z$ exists but is unknown to the algorithm, we can still improve $X$ since the set of $X$-endpoints of $\mathcal{P}$ also form a redundant set in $X$.

We introduce both notions to keep consistency with the earlier work [30] but in fact we will be interested only in weak coverings. Following the example above, the set $Z = \chi(t)$ is weakly $(\mathcal{H}, k + 1)$-separable but not necessarily $(\mathcal{H}, k + 1)$-separable.

Next, we introduce the notion of redundancy for solutions to $\mathcal{H}$-DELETION.

▶ **Definition 10.** *For an $\mathcal{H}$-deletion set $X$ in $G$ we say that a subset $X' \subseteq X$ is* redundant *in $X$ if there exists a set $X'' \subseteq V(G)$ smaller than $|X'|$ such that $(X \setminus X') \cup X''$ is also an $\mathcal{H}$-deletion set in $G$.*

We remark that redundancy has been studied in the context of local-search strategies (cf. [27, 28]). It is known that for VERTEX COVER finding a redundant subset $X'$ in a solution $X$ is FPT in graphs of bounded local treewidth but W[1]-hard in general, when parameterized by the size of $|X'|$ [22]. However, when $X'$ is given, one can easily check whether it is redundant using an algorithm for $\mathcal{H}$-DELETION parameterized by the solution size, due to the following observation.

▶ **Observation 11.** *Let $X$ be an $\mathcal{H}$-deletion set in a graph $G$. A subset $X' \subseteq X$ is redundant in $X$ if and only if the graph $G - (X \setminus X')$ has an $\mathcal{H}$-deletion set smaller than $|X'|$.*

An important observation is that when $X' \subseteq X$ of size at least $\ell + 1$ is weakly $(\mathcal{H}, \ell)$-separable, then it is redundant in $X$ by a simple exchange argument. This fact has been already leveraged in previous work [1, 30] when analyzing the structure of minimum-size $\mathcal{H}$-deletion sets, which clearly cannot contain any redundant subsets. We exploit it in a different context, to prove that if there is a large flow between an $\mathcal{H}$-deletion set $X$ and a weakly $(\mathcal{H}, \ell)$-separable set $Z$, then $X$ has a redundant subset. Subsequently, we will show that this redundant subset can efficiently be detected.

▶ **Lemma 12.** *Let $\mathcal{H}$ be a hereditary and union-closed class of graphs. Consider a graph $G$, an $\mathcal{H}$-deletion set $X$ in $G$, and a weakly $(\mathcal{H}, \ell)$-separable set $Z \subseteq V(G)$. Suppose that there exists a subset $X' \subseteq X$ of size $2\ell + 1$ such that $\lambda_G(Z, X') = 2\ell + 1$. Then $X'$ is redundant in $X$.*

**Proof.** Let $(C, S)$ be an $(\mathcal{H}, \ell)$ separation in $G$ with $Z \subseteq C \cup S$. From the definition of an $(\mathcal{H}, \ell)$-separation, we have $G[C] \in \mathcal{H}$ while $N_G(C) \subseteq S$ and $|S| \leq \ell$.

By Menger's theorem, the cardinality of a maximum packing of vertex-disjoint $(Z, X')$-paths equals $\lambda_G(Z, X')$. Hence there exists a family $\mathcal{P} = \{P_1, \ldots, P_{2\ell+1}\}$ of vertex-disjoint paths, each of which connects a unique vertex $z_i \in Z$ to a unique vertex $x_i \in X'$ (possibly $x_i = z_i$). At most $|S|$ of these paths intersect the separator $S$ of the $(\mathcal{H}, \ell)$-separation (see Figure 1). Let $X'' = \{x_i \mid P_i \cap S = \emptyset\}$ denote the $X'$-endpoints of those paths not intersecting $S$, and let $\mathcal{P}'$ be the corresponding paths. Each path $P_i$ in $\mathcal{P}'$ is disjoint from $S$ and has an endpoint $z_i \in Z$. Since $Z \subseteq C \cup S$, the $z_i$ endpoint belongs to $C$. As $N_G(C) \subseteq S$ and $P_i$ does not intersect $S$, the other endpoint $x_i$ also belongs to $C$. Hence all vertices of $X''$ belong to $C$, and there are at least $2\ell + 1 - \ell = \ell + 1$ of them.

Let $X^* := (X \setminus X'') \cup S$, and observe that $|X^*| < |X|$ since $|X''| \geq \ell + 1$ while $|S| \leq \ell$. We prove that $G - X^* \in \mathcal{H}$, by showing that $S$ is an $\mathcal{H}$-deletion set in $G - (X \setminus X'')$. Since $\mathcal{H}$ is union-closed, it suffices to argue that each connected component $H$ of $G - ((X \setminus X'') \cup S)$ belongs to $\mathcal{H}$. If $H$ contains no vertex of $X''$, then $H$ is an induced subgraph of $G - X \in \mathcal{H}$ and therefore $H \in \mathcal{H}$ since the graph class is hereditary. If $H$ contains a vertex of $X'' \subseteq C$, then the component $H$ is an induced subgraph of $G[C]$ since $N_G(C) \subseteq S$ is part of the set $X^*$. Hence $H$ is an induced subgraph of $G[C] \in \mathcal{H}$, which implies $H \in \mathcal{H}$ as $\mathcal{H}$ is hereditary. This shows that $X^*$ is indeed an $\mathcal{H}$-deletion set.

Since $(X \setminus X'') \cup S$ is an $\mathcal{H}$-deletion set smaller than $X$, the set $X''$ is redundant in $X$. As $X' \supseteq X''$, it follows that $X'$ is redundant as well. ◀

## 3.2 The win/win strategy

The classic 4-approximation algorithm for computing a (standard) tree decomposition is based on the existence of balanced separators in graphs of bounded treewidth. In a graph $G$ of treewidth $\leq k$, any set $S$ of $3k + 4$ vertices can by partitioned into $S = S_A \cup S_B$ in such a way that $|S_A|, |S_B| \leq 2k + 2$ and $\lambda_G(S_A, S_B) \leq k + 1$ [16, Corollary 7.21]. This is not always possible if we only have a bound on $\mathcal{H}$-treewidth $\mathbf{tw}_{\mathcal{H}}(G) \leq k$ because a large subset $S'$ of $S$ might lie in a single well-connected base component of a tree $\mathcal{H}$-decomposition, i.e., a base component whose standard treewidth is large. But then $S'$ is weakly $(\mathcal{H}, k + 1)$-separable, which can also be exploited when constructing a decomposition. We show that this is in fact the only scenario in which we cannot split $S$ in a balanced way.

▶ **Lemma 13.** *Let $\mathcal{H}$ be a hereditary and union-closed class of graphs. Let $G$ be a graph with $\boldsymbol{tw}_{\mathcal{H}}(G) \leq k$. For any set $S \subseteq V(G)$ of size $3k + 4$, at least one of the following holds.*

1. *There is a partition $S = S_A \cup S_B$ such that $|S_A|, |S_B| \leq 2k + 2$ and $\lambda_G(S_A, S_B) \leq k + 1$.*
2. *There is a set $S' \subseteq S$ of size $2k + 3$ which is weakly $(\mathcal{H}, k + 1)$-separable.*

**Proof.** Consider an optimal tree $\mathcal{H}$-decomposition $(T, \chi, L)$ of $G$, so that $|\chi(t) \setminus L| \leq k + 1$ for each $t \in V(T)$. Let $r \in V(T)$ be its root. We start by showing that (2) holds if some leaf bag of the decomposition contains $2k + 3$ vertices from $S$.

So suppose there exists a leaf $t \in V(T)$ with $|\chi(t) \cap S| \geq 2k + 3$, and let $S' \subseteq \chi(t) \cap S$ be an arbitrary subset of size exactly $2k + 3$. Observation 8 ensures that $(C^* := \chi(t) \cap L, S^* := \chi(t) \setminus L)$ is an $(\mathcal{H}, k + 1)$-separation, which weakly covers $\chi(t)$ and therefore $S'$. Hence (2) holds.

In the remainder, it suffices to show that (1) holds when there is no leaf $t \in V(T)$ with $|\chi(t) \cap S| \geq 2k + 3$. Pick a deepest node $t^*$ in the rooted tree $T$ for which $|S \cap \chi(T_{t^*})| \geq 2k + 3$. Then $t^*$ is not a leaf since the previous case did not apply, so by definition of tree $\mathcal{H}$-decomposition we have $\chi(t^*) \cap L = \emptyset$. Let $D_1, \ldots, D_p$ be the connected components of $G - \chi(t^*)$. Since the pair $(T, \chi)$ satisfies all properties of a standard tree decomposition, the bag $\chi(t^*)$ is a separator in $G$ so that for each component $D_i$, there is a single tree $T^i$ in the unrooted forest $T - t^*$ such that $T^i$ contains all nodes whose bags contain some $v \in V(D_i)$; see for example [46, (2.3)].

The choice of $t^*$ ensures that $|V(D_i) \cap S| < 2k+3$ for all $i \in [p]$: when vertices of $D_i$ are contained in bags of a tree rooted at a child of $t^*$ this follows from the fact that $t^*$ is a deepest node for which $|S \cap \chi(T_{t^*})| \geq 2k+3$; when vertices of $D_i$ are contained in the tree $T^i$ of $T - t^*$ having the parent of $t^*$, this follows from the fact that $\chi(T_{t^*})$ contains at least $2k+3$ vertices from $S$, none of which appear in $D_i$ since a vertex occurring in $\chi(T_{t^*})$ and in a bag outside $T_{t^*}$, is contained in $\chi(t^*)$ and therefore part of the separator $\chi(t^*)$ used to obtain the component $D_i$. Hence none of the vertices of $S \cap \chi(t^*)$ can appear in $D_i$, which means there are at most $|S| - (2k+3) \leq k+1$ vertices in $V(D_i) \cap S$.

Since $|S| = 3k+4$ and no component $D_i$ contains at least $2k+3$ vertices from $S$, the components can be partitioned into two parts $\mathcal{D}_1, \mathcal{D}_2$ such that $\sum_{D_i \in \mathcal{D}_j} |V(D_i) \cap S| \leq 2k+2$ for each $j \in \{1,2\}$. If some component contains at least $k+2$ vertices from $S$, then that component is a part by itself, ensuring the remainder has at most $3k+4-(k+2) \leq 2k+2$ vertices from $S$; if no component contains at least $k+2$ vertices from $S$, then any inclusion-minimal subset of components having at least $k+2$ vertices from $S$ has at most $2k+2$ of them.

Define $S'_A := \bigcup_{D_i \in \mathcal{D}_1} V(D_i) \cap S$ and $S'_B := \bigcup_{D_i \in \mathcal{D}_2} V(D_i) \cap S$, and assume without loss of generality that $|S'_A| \geq |S'_B|$. Note that $|S'_A \cup S'_B| = |S \setminus \chi(t^*)| \geq 2k+3$, so that the larger side $S'_A$ contains at least $k+2$ vertices. To turn $S'_A, S'_B$ into the desired partition of $S$, it suffices to take $S_A = S'_A$ and $S_B = S'_B \cup (\chi(t^*) \cap S) = S \setminus S_A$. It is clear that $|S_A| = |S'_A| \geq k+2$, while $|S_B| = |S| - |S_A| \geq 3k+4-(2k+2) \geq k+2$. The fact that $|S_A|, |S_B| \geq k+2$ while they partition $S$ with $|S| = 3k+4$ implies $|S_A|, |S_B| \leq 2k+2$ as desired. Since $\chi(t^*)$ separates $S'_A$ from $S'_B$, it separates $S_A$ from $S_B$ and we have $\lambda_G(S_A, S_B) \leq |\chi(t^*)| = |\chi(t^*) \setminus L| \leq k+1$. ◀

We can now translate the last two lemmas into an algorithmic statement, which will be used as a subroutine in the main algorithm. When $\mathbf{tw}_{\mathcal{H}}(G) \leq k$ and $S \subseteq V(G)$ is of size $3k+4$, then we can either split it in a balanced way, split off a base component, or detect a redundancy in a given $\mathcal{H}$-deletion set and reduce its size. Each of these outcomes will guarantee some progress for the task of constructing a tree $\mathcal{H}$-decomposition.

▶ **Lemma 14.** *Let $\mathcal{H}$ be a hereditary and union-closed class of graphs. There is an algorithm that, using oracle-access to an algorithm $\mathcal{A}$ for $\mathcal{H}$-DELETION, takes as input an $n$-vertex $m$-edge graph $G$, integer $k$, $\mathcal{H}$-deletion set $X$ in $G$, and a set $S \subseteq V(G)$ of size $3k+4$, runs in time $\mathcal{O}(8^k \cdot k(n+m))$ and polynomial space, makes $\mathcal{O}(8^k)$ calls to $\mathcal{A}$ on induced subgraphs of $G$ and parameter $2k+2$, and terminates with one of the following outcomes.*

1. *A partition $S = S_A \cup S_B$ and a separation $(A, B)$ in $G$ are returned, such that $S_A \subseteq A$, $S_B \subseteq B$, $|S_A| \leq 2k+2$, $|S_B| \leq 2k+2$, and $|A \cap B| \leq k+1$.*
2. *A subset $S' \subseteq S$ and a separation $(A, B)$ in $G$ are returned, such that $S' \subseteq A$, $X \subseteq B$, $|S'| = 2k+3$, and $|A \cap B| \leq 2k+2$. (This implies that $G[A \setminus B] \in \mathcal{H}$.)*
3. *An $\mathcal{H}$-deletion set $X'$ in $G$ is returned, that is smaller than $X$.*
4. *The algorithm correctly concludes that $\boldsymbol{tw}_{\mathcal{H}}(G) > k$.*

**Proof.** The algorithm starts by trying to reach the first outcome. For each partition $S_A \cup S_B$ of $S$ in which both parts have at most $2k+2$ vertices, it performs at most $k+2$ iterations of the Fold-Fulkerson algorithm to test whether $\lambda_G(S_A, S_B) \leq k+1$. If so, then the algorithm outputs a corresponding separation $(A, B)$ in $G$ with $S_A \subseteq A$, $S_B \subseteq B$, and $|A \cap B| = \lambda_G(S_A, S_B) \leq k+1$. By Theorem 4, this can be done in time $\mathcal{O}(k(n+m))$.

Next, the algorithm attempts to reach the second outcome. For each subset $S' \subseteq S$ of size $2k+3$, it performs at most $2k+3$ iterations of the Ford-Fulkerson algorithm to test whether $\lambda_G(S', X) \leq 2k+2$. If so, the algorithm extracts a corresponding separation $(A, B)$ with $S' \subseteq A$, $X \subseteq B$, and $|A \cap B| \leq 2k+2$, and outputs it.

If the algorithm has not terminated so far, it will reach the third or fourth outcome. It proceeds as follows.

1. For each subset $S' \subseteq S$ of size $2k+3$, we have $\lambda_G(S', X) > 2k+2$ since we could not reach the second outcome. As $|S'| = 2k+3$ this implies $\lambda_G(S', X) = 2k+3$. By Menger's theorem, there is a packing $\mathcal{P}_{S'}$ of $2k+3$ vertex-disjoint $(S', X)$-paths, and such a packing can be extracted from the final stage of the Ford-Fulkerson computation.
2. Let $X'_{S'} \subseteq X$ be the endpoints in the set $X$ of the paths $\mathcal{P}_{S'}$, so that $|X'_{S'}| = |S'| = 2k+3$.
3. We invoke algorithm $\mathcal{A}$ on the graph $G - (X \setminus X'_{S'})$ and parameter value $2k+2$, to find a minimum-size $\mathcal{H}$-deletion set in $G - (X \setminus X'_{S'})$ or conclude that such a set has size more than $2k+2$. If $\mathcal{A}$ returns a solution $Y$ of size at most $2k+2$, then $(X \setminus X'_{S'}) \cup Y$ is an $\mathcal{H}$-deletion set in $G$ smaller than $X$ and we return it as the third outcome.

If none of the preceding steps for any $S' \subseteq S$ of size $2k+3$ caused the algorithm to give an output, then we conclude that $\mathbf{tw}_{\mathcal{H}}(G) > k$ and terminate.

**Correctness.** We proceed to argue for correctness of the algorithm. It is clear that if the algorithm terminates with one of the first three outcomes, then its output is correct. We proceed to show that if $\mathbf{tw}_{\mathcal{H}}(G) \leq k$, then it will indeed terminate in one of those outcomes. So assume $\mathbf{tw}_{\mathcal{H}}(G) \leq k$, which means we may apply Lemma 13 to $S$ and $G$. If Case 1 of Lemma 13 holds, then the algorithm will detect the corresponding separation in the first phase of the algorithm and terminate with a suitable separation. So assume Case 2 holds, so that there is a set $S' \subseteq S$ of size $2k+3$ which is weakly $(\mathcal{H}, k+1)$-separable. Since the set $S'$ is a candidate for reaching the second outcome, if that outcome is not reached we have $\lambda_G(S', X) > 2k+2$ and hence $\lambda_G(S', X) = 2k+3 = |S'|$. Consider the family of $(S', X)$-paths $\mathcal{P}_{S'}$ constructed by the algorithm for this choice of $S'$ and let $X'_{S'}$ be their endpoints in $X$. The paths $\mathcal{P}_{S'}$ show that $\lambda_G(S', X'_{S'}) = |S'| = |X'_{S'}| = 2k+3$. Now we can apply Lemma 12 for $\ell = k+1$ to infer that $X'_{S'}$ is redundant in $X$, which implies that $G - (X \setminus X'_{S'})$ has an $\mathcal{H}$-deletion set smaller than $|X'_{S'}| = 2k+3$. Hence algorithm $\mathcal{A}$ outputs an $\mathcal{H}$-deletion set smaller than $|X'_{S'}|$ and the algorithm terminates with the third outcome.

Since the algorithm reaches one the first three outcomes when $\mathbf{tw}_{\mathcal{H}}(G) \leq k$, the algorithm is correct when it reaches the last outcome.

**Running time and oracle calls.** Each of the three phases of the algorithm consist of enumerating subsets $S' \subseteq S$, of which there are $2^{|S|} \leq 2^{3k+4} = \mathcal{O}(8^k)$. For each such set $S'$, the algorithm performs $\mathcal{O}(k)$ rounds of the Ford-Fulkerson algorithm in time $\mathcal{O}(k(n+m))$. In the last phase, the algorithm additionally invokes $\mathcal{A}$ on an induced subgraph of $G$ for each $S'$ to find an $\mathcal{H}$-deletion set of size at most $2k+2$ if one exists. It follows that the running time of the algorithm (not accounting for the time spent by $\mathcal{A}$) is $\mathcal{O}(8^k \cdot k(n+m))$. The space usage is easily seen to be polynomial in the input size since the algorithm is iterative. This concludes the proof of Lemma 14. ◀

## 3.3 The decomposition algorithm

We retrace the proof of [16, Theorem 7.18] which gives the classic algorithm for approximating (standard) treewidth. Consider sets $S \subseteq W \subseteq V(G)$ such that $\partial_G(W) \subseteq S$ and $|S| = 3k+4$; we aim to construct a tree decomposition of $G[W]$ which contains $S$ in its root bag. We can consider all ways to partition $S$ into $S_A \cup S_B$ such that $|S_A|, |S_B| \leq 2k+2$ and compute a minimum $(S_A, S_B)$-separator. Since $|S| = 3k+4$, there are $2^{3k+4} = \mathcal{O}(8^k)$ such partitions.

When $\mathbf{tw}(G) \leq k$, we are guaranteed that for some partition $S = S_A \cup S_B$ we will find a separator in $G[W]$ of size $\leq k+1$ which yields the separation $(A_W, B_W)$ in $G[W]$ satisfying $S_A \subseteq A_W$, $S_B \subseteq B_W$, and $|A_W \cap B_W| \leq k+1$. Then the boundary $\partial_G(A_W)$ is contained in $S_A \cup (A_W \cap B_W)$, and similarly $\partial_G(B_W) \subseteq S_B \cup (A_W \cap B_W)$. We create instances $(A_W, S_A \cup (A_W \cap B_W))$ and $(B_W, S_B \cup (A_W \cap B_W))$ to be solved recursively, analogously as $(W, S)$. Note that each of the sets $S_A \cup (A_W \cap B_W)$, $S_B \cup (A_W \cap B_W)$ has less than $3k+4$ vertices, so we can augment each of them with one more vertex before making the recursive call while preserving the size invariant. This step ensures that the recursion tree has at most $|V(G)|$ nodes. After computing tree decompositions for $G[A]$ and $G[B]$ we merge them by creating a new root with a bag $S \cup (A_W \cap B_W)$ of size at most $4k+5$. Hence, we are able to construct a tree decomposition of width $4k+4$ assuming that one of width $k$ exists.

There are two differences between the outlined algorithm and ours, while the recursive scheme stays the same. First, due to scenario (2) in Lemma 14 we need to handle the cases where we can directly create a base component containing at least $2k+3$ vertices from $S$. The lower bound $2k+3$ is greater than the separator size $2k+2$ so we will move on to a subproblem where $S$ is significantly smaller. We need to include the separator of size $2k+2$ in the root bag, together with $S$, so we obtain a slightly weaker bound on the maximum bag size, that is $5k+6$. Next, due to scenario (3) we might not make direct progress in the recursive scheme but instead we reduce the size of an $\mathcal{H}$-deletion set $X$ that we maintain (which initially contains all vertices). This situation can happen at most $|V(G)|$ many times, so eventually we will reach outcome (1) or (2).

The approach sketched above leads to a proof of Theorem 1. The details are deferred to the full version [31] due to space restrictions. Apart from carefully combining the ingredients collected so far, in the proof we take care to optimize the number of calls to the $\mathcal{H}$-DELETION oracle, leading to the clean bound of $\mathcal{O}(8^k n)$ oracle calls advocated in the introduction.

## 4 Conclusion

We contributed to the algorithmic theory of hybrid graph parameterizations, by showing how a 5-approximation to $\mathbf{tw}_{\mathcal{H}}$ can be obtained using an algorithm for the solution-size parameterization of $\mathcal{H}$-DELETION as a black box. This makes the step of computing a tree $\mathcal{H}$-decomposition now essentially as fast as that of solving $\mathcal{H}$-DELETION parameterized by solution size. Our new decomposition algorithm combines with existing algorithms to solve $\mathcal{H}$-DELETION on a given tree $\mathcal{H}$-decomposition, to deliver algorithms that solve $\mathcal{H}$-DELETION parameterized by $\mathbf{tw}_{\mathcal{H}}$. For Odd Cycle Transversal and Vertex Planarization, the parameter dependence of the resulting algorithm is equal to the worst of the parameter dependencies of the solution-size and treewidth-parameterizations. We believe that this is not a coincidence, and offer the following conjecture.

▶ **Conjecture 15.** *Let $\mathcal{H}$ be a hereditary and union-closed graph class. If $\mathcal{H}$-DELETION can be solved in time $f(s) \cdot n^{\mathcal{O}(1)}$ parameterized by solution size $s$, and in time $h(w) \cdot n^{\mathcal{O}(1)}$ parameterized by treewidth $w$, then $\mathcal{H}$-DELETION can be solved in time $(f(\mathcal{O}(k)) + h(\mathcal{O}(k))) \cdot n^{\mathcal{O}(1)}$ parameterized by $\mathcal{H}$-treewidth $k$.*

The conjecture is a significant strengthening of the equivalence, with respect to non-uniform fixed-parameter tractability, between solving $\mathcal{H}$-DELETION parameterized by solution size and computing $\mathbf{tw}_{\mathcal{H}}$ given by Agrawal et al. [1]. It essentially states that there is no *price of generality* to pay for using the hybrid parameterization by $\mathbf{tw}_{\mathcal{H}}$. After three decades in which the field of parameterized complexity has focused on parameterizations by solution size, this would lead to a substantial shift of perspective. We believe Theorem 1 is an important ingredient in this direction.

To understand the relative power of the parameterizations by solution size, treewidth, and $\mathcal{H}$-treewidth, the remaining bottleneck lies in using the tree $\mathcal{H}$-decomposition to compute a minimum $\mathcal{H}$-deletion set. Can the latter be done as efficiently when using a tree $\mathcal{H}$-decomposition as when using a standard tree decomposition? For problems like Odd Cycle Transversal and Vertex Planarization, this is indeed the case. But when the current-best dynamic-programming algorithm over a tree decomposition uses advanced techniques, it is currently not clear how to lift such an algorithm to work on a tree $\mathcal{H}$-decomposition. Can $\mathcal{H}$-deletion for $\mathcal{H}$ the class of interval graphs be solved in time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ parameterized by $\mathbf{tw}_{\mathcal{H}}$? Such a running time can be obtained for the parameterization by treewidth by adapting the approach of Saitoh, Yoshinaka, and Bodlaender [48].

While we have not touched on the subject here, we expect our ideas to also be applicable when $\mathcal{H}$ is a *scattered graph class*, i.e., when $\mathcal{H}$ consists of graphs where each connected component is contained in one of a finite number of graph classes $\mathcal{H}_1, \ldots, \mathcal{H}_t$. It is known [30] that, when Vertex Cover can be solved in polynomial time on each graph class $\mathcal{H}_i$, then Vertex Cover is FPT parameterized by the width of a given tree $\mathcal{H}$-decomposition. We expect that Theorem 1 can be generalized to work with scattered graph classes $\mathcal{H}$, as long as there is an oracle to solve $\mathcal{H}_i$-deletion parameterized by solution size for each individual class $\mathcal{H}_i$. To accommodate this setting, the algorithm maintains an $\mathcal{H}_i$-deletion set $X_i$ for *each* graph class $\mathcal{H}_i$. A step of the decomposition algorithm then either consists of finding a balanced separation of $S$, splitting off a base component, or improving *one of the* deletion sets $X_i$ (which can occur only $t \cdot |V(G)|$ times).

The decomposition algorithm we presented has an approximation factor of 5. It may be possible to obtain a smaller approximation ratio at the expense of a worse base of the exponent, by repeatedly splitting large bags [5, 35, 36]. For obtaining single-exponential $\mathcal{H}$-deletion algorithms, the advantage of the improved approximation factor would be immediately lost due to the increased running time and therefore we did not pursue this direction.

A final direction for future work concerns the optimization of the polynomial part of the running time. For standard treewidth, a 2-approximation can be computed in time $2^{\mathcal{O}(k)} \cdot n$ [35], which was obtained after a long series of improvements (cf. [8, Table 1]) on both the approximation factor and dependence on $n$. Can a constant-factor approximation to $\mathcal{H}$-treewidth be computed in time $2^{\mathcal{O}(k)} \cdot (n+m)$ for graph classes $\mathcal{H}$ like bipartite graphs?

## References


1 Akanksha Agrawal, Lawqueen Kanesh, Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Deleting, eliminating and decomposing to hereditary classes are all FPT-equivalent. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022*, pages 1976–2004. SIAM, 2022. doi:10.1137/1.9781611977073.79.

2 Akanksha Agrawal and M. S. Ramanujan. Distance from triviality 2.0: Hybrid parameterizations. In Cristina Bazgan and Henning Fernau, editors, *Combinatorial Algorithms – 33rd International Workshop, IWOCA 2022, Trier, Germany, June 7-9, 2022, Proceedings*, volume 13270 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2022. doi:10.1007/978-3-031-06678-8_1.

3 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

4 Mahdi Belbasi and Martin Fürer. An improvement of reed's treewidth approximation. *J. Graph Algorithms Appl.*, 26(2):257–282, 2022. doi:10.7155/jgaa.00593.

5 Patrick Bellenbaum and Reinhard Diestel. Two short proofs concerning tree-decompositions. *Comb. Probab. Comput.*, 11(6):541–547, 2002. doi:10.1017/S0963548302005369.

6 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

7 Hans L. Bodlaender. A partial $k$-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

8 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016. doi:10.1137/130947374.

9 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

10 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations I. Upper bounds. *Inf. Comput.*, 208(3):259–275, 2010. doi:10.1016/j.ic.2009.03.008.

11 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations II. Lower bounds. *Inf. Comput.*, 209(7):1103–1119, 2011. doi:10.1016/j.ic.2011.04.003.

12 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016. doi:10.1007/s00453-015-0045-3.

13 Jannis Bulian and Anuj Dawar. Fixed-parameter tractable distances to sparse graph classes. *Algorithmica*, 79(1):139–158, 2017. doi:10.1007/s00453-016-0235-7.

14 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/j.tcs.2010.06.026.

15 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. doi:10.1017/CBO9780511977619.

16 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

17 Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. The first parameterized algorithms and computational experiments challenge. In Jiong Guo and Danny Hermelin, editors, *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*, volume 63 of *LIPIcs*, pages 30:1–30:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.30.

18 Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2017.30.

19 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

20 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

21 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 42:1–42:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.42.

22 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *J. Comput. Syst. Sci.*, 78(3):707–719, 2012. doi:10.1016/j.jcss.2011.10.003.

23 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

24 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: on the price of generality. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 825–834. SIAM, 2009. doi:10.1137/1.9781611973068.90.

25 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM J. Comput.*, 39(5):1941–1956, 2010. doi:10.1137/080742270.

26 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM J. Comput.*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

27 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013. doi:10.1007/s00453-012-9685-8.

28 Jiong Guo, Danny Hermelin, and Christian Komusiewicz. Local search for string problems: Brute-force is essentially optimal. *Theor. Comput. Sci.*, 525:30–41, 2014. doi:10.1016/j.tcs.2013.05.006.

29 Bart M. P. Jansen and Jari J. H. de Kroon. FPT algorithms to compute the elimination distance to bipartite graphs and more. In Lukasz Kowalik, Michal Pilipczuk, and Pawel Rzazewski, editors, *Graph-Theoretic Concepts in Computer Science – 47th International Workshop, WG 2021, Warsaw, Poland, Revised Selected Papers*, volume 12911 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 2021. doi:10.1007/978-3-030-86838-3_6.

30 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 1757–1769, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451068.

31 Bart M. P. Jansen, Jari J. H. de Kroon, and Michal Włodarczyk. 5-approximation for $H$-treewidth essentially as fast as $H$-deletion parameterized by solution size. *CoRR*, abs/2306.17065, 2023. arXiv:2306.17065.

32 Bart M. P. Jansen, Jari J. H. de Kroon, and Michał Włodarczyk. Vertex deletion parameterized by elimination distance and even less. *CoRR*, abs/2105.04660, 2021. URL: https://arxiv.org/abs/2105.04660.

33 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1802–1811. SIAM, 2014. doi:10.1137/1.9781611973402.130.

34 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 639–648. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.45.

35 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 184–192. IEEE, 2021. doi:10.1109/FOCS52979.2021.00026.

36 Tuukka Korhonen and Daniel Lokshtanov. An improved parameterized algorithm for treewidth. *CoRR*, abs/2211.07154, 2022. doi:10.48550/arXiv.2211.07154.

37 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

38 Dániel Marx. Four shorts stories on surprising algorithmic uses of treewidth. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 129–144. Springer, 2020. doi:10.1007/978-3-030-42071-0_10.

39 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

40 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

41 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

42 Marcin Pilipczuk. A tight lower bound for vertex planarization on graphs of bounded treewidth. *Discret. Appl. Math.*, 231:211–216, 2017. doi:10.1016/j.dam.2016.05.019.

43 Bruce A. Reed. Finding approximate separators and computing tree width quickly. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 221–228. ACM, 1992. doi:10.1145/129712.129734.

44 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

45 N. Robertson and P.D. Seymour. Graph minors XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

46 Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

47 Neil Robertson and Paul D. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48(2):227–254, 1990. doi:10.1016/0095-8956(90)90120-O.

48 Toshiki Saitoh, Ryo Yoshinaka, and Hans L. Bodlaender. Fixed-treewidth-efficient algorithms for edge-deletion to interval graph classes. In Ryuhei Uehara, Seok-Hee Hong, and Subhas C. Nandy, editors, *WALCOM: Algorithms and Computation – 15th International Conference and Workshops, WALCOM 2021, Yangon, Myanmar, February 28 – March 2, 2021, Proceedings*, volume 12635 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2021. doi:10.1007/978-3-030-68211-8_12.

49 A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

# The Unweighted and Weighted Reverse Shortest Path Problem for Disk Graphs

**Haim Kaplan**
School of Computer Science, Tel Aviv University, Israel

**Matthew J. Katz**
Department of Computer Science, Ben Gurion University, Beer-Sheva, Israel

**Rachel Saban**
Department of Computer Science, Ben Gurion University, Beer-Sheva, Israel

**Micha Sharir**
School of Computer Science, Tel Aviv University, Israel

## Abstract

We study the reverse shortest path problem on disk graphs in the plane. In this problem we consider the proximity graph of a set of $n$ disks in the plane of arbitrary radii: In this graph two disks are connected if the distance between them is at most some threshold parameter $r$. The case of intersection graphs is a special case with $r = 0$. We give an algorithm that, given a target length $k$, computes the smallest value of $r$ for which there is a path of length at most $k$ between some given pair of disks in the proximity graph. Our algorithm runs in $O^*(n^{5/4})$ randomized expected time, which improves to $O^*(n^{6/5})$ for unit disk graphs, where all the disks have the same radius.[1] Our technique is robust and can be applied to many variants of the problem. One significant variant is the case of weighted proximity graphs, where edges are assigned real weights equal to the distance between the disks or between their centers, and $k$ is replaced by a target weight $w$. In other variants, we want to optimize a parameter different from $r$, such as a scale factor of the radii of the disks.

The main technique for the decision version of the problem (determining whether the graph with a given $r$ has the desired property) is based on efficient implementations of BFS (for the unweighted case) and of Dijkstra's algorithm (for the weighted case), using efficient data structures for maintaining the bichromatic closest pair for certain bicliques and several distance functions. The optimization problem is then solved by combining the resulting decision procedure with enhanced variants of the interval shrinking and bifurcation technique of [4].

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

**Keywords and phrases** Computational geometry, geometric optimization, disk graphs, BFS, Dijkstra's algorithm, reverse shortest path

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.67

**Related Version** *Full Version*: https://doi.org/10.48550/arXiv.2307.14663

**Funding** *Haim Kaplan*: Work partially supported by Grant 1595/19 from the Israel Science Foundation and by the Blavatnik Research Foundation.
*Matthew J. Katz*: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
*Micha Sharir*: Work partially supported by Grant 260/18 from the Israel Science Foundation.

## 1 Introduction

In this paper we study the *reverse shortest path problem* (RSP for short) on graphs defined by disks in the plane.

[1] In this paper the $O^*(\cdot)$ notation hides subpolynomial factors.




In the simplest variant of this problem, we are given a set $P$ of $n$ points in the plane, two designated points $s, t \in P$, a real parameter $r > 0$ and an integer $k < n$. Define $G_r$ to be the graph $(P, E_r)$, where the edges of $E_r$ are all the pairs $(p, q)$ such that $\|p - q\| \leq 2r$. This is also the intersection graph of the disks of radius $r$ centered at the points of $P$. In the decision version of the RSP problem we want to determine whether $G_r$ contains a path from $s$ to $t$ with at most $k$ edges. In the optimization version, which is the reverse shortest path problem itself, we wish to find the smallest value $r^*$ for which $G_{r^*}$ has this property. Both versions (decision and optimization) of this problem have received considerable attention during the past decade [5, 7, 14].

**Our contributions.** We give an algorithm for this problem that runs in $O^*(n^{6/5})$ randomized expected time (where the $O^*(\cdot)$ notation hides subpolynomial factors). This improves the recent $O^*(n^{5/4})$-time solution of Wang and Zhao [14]. In fact we study the RSP problem in a much more general context that involves a variety of intersection or proximity graphs on a finite set of arbitrary disks in the plane, for which nontrivial bounds were not known prior to this work.

We first consider unweighted disk graphs in Section 3. In this setup, we have a set $\mathcal{D}$ of $n$ disks in the plane of arbitrary radii. Each disk $D \in \mathcal{D}$ is specified by its center $c_D$ and radius $\rho_D$. For a given parameter $r \geq 0$ we define the *proximity graph* $G_r$ by adding an edge between disks $D$ and $D'$ if the distance between them, $\mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'}$, is at most $r$. The case $r = 0$ is of special interest and gives the intersection graph of the disks.[2]

For the decision version of this RSP problem we obtain an algorithm that runs in $O(n \log^4 n)$ time, and for the optimization version an algorithm that runs in $O^*(n^{5/4})$ randomized expected time. Our technique generalizes to other versions of the optimization problem. For example, we can consider the intersection graph of the disks ($r = 0$) and ask for the smallest scaling factor of the radii of all disks, either by a common additive term or by a common multiplicative factor, that would make the graph contain a path of at most $k$ edges between a designated pair of source and target disks $D_s$ and $D_t$.

In Section 4 we generalize our results further to weighted versions of the proximity graph $G_r$. We consider two natural weight functions for the edges. The first sets the weight of an edge $(D, D')$ to be the distance $\|c_D - c_{D'}\|$ between the centers of the disks, and the second sets the weight to be the distance $\mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'}$ between the disks. We solve the decision problem on such weighted disk graphs, in which we want to determine whether the shortest path in $G_r$ from $D_s$ to $D_t$ is of length at most $w$, for some specified real threshold $w$, by a careful implementation of Dijkstra's algorithm (using a dynamic bichromatic closest pair structure, see below) in $O(n \log^4 n)$ or $O(n \log^6 n)$ time, depending on the type of edge weights. The optimization RSP problem is then solved in $O^*(n^{5/4})$ randomized expected time. For weighted unit disk graphs we still get the better bound of $O^*(n^{6/5})$ time for the optimization problem.

Our decision algorithms rely on rather complex dynamic data structures (see below), which should be avoided, if possible, from a practical point of view. Indeed, for unit disk graphs, there exist simpler and slightly more efficient implementations of BFS and Dijkstra's algorithm, in the unweighted and weighted cases, respectively [5, 7, 13]. However, as explained in the remarks following Theorems 2 and 7, we cannot use them in conjunction with our optimization technique, for certain technical reasons. We thus present in the full version of this paper alternative implementations, based on the known grid-based techniques, which satisfy our requirements and are arguably somewhat simpler.

[2] One technical difference is that when considering proximity graphs, it is customary, although not obligatory, to assume that the disks are pairwise disjoint, which is certainly not an assumption that one would make for intersection graphs.

**Our techniques.** We achieve our results by carefully combining three main technical ingredients. The first is an efficient "serial" implementation of parametric search, using what we call *interval shrinking* and *bifurcation* procedures. This technique was first used by Ben Avraham et al. [4] for solving problems involving the discrete and semi-discrete Fréchet distance with shortcuts. Here we apply a somewhat modified variant of it in the rather different context of our RSP problem, exposing its potential to be useful for a wide range of other problems as well.

We remark, that using this technique requires that the decision procedure access the parameter $r$ to be optimized via comparisons only, whose outcome depends on the relation between the optimal $r^*$ and certain critical values (which in our case turn out to be additively weighted inter-point distances between the centers of the disks), on which we can apply the interval shrinking procedure in an efficient manner. We review this technique in Section 2.

The second ingredient is a dynamic nearest neighbor and a dynamic bichromatic closest pair data structures for additively weighted Euclidean distances. Such structures with polylogarithmic time per update and access were recently developed by Kaplan et al. [8] and Liu [11].

The third ingredient that we use for the weighted versions of the problem is a technique that combines nearest neighbor data structures for two different distance functions. Specifically, given a (dynamic) nearest neighbor data structure for a distance function $d_1$, and a (dynamic) nearest neighbor data structure for a distance function $d_2$, we show how we can get a (dynamic) data structure that can answer constrained nearest neighbor queries of the form: find the closest point to a query $q$ according to the distance function $d_1$ among all points whose distance to $q$ according to $d_2$ is at most some threshold $r$ (which is part of the query).

**Previous work.** The decision problem in the unweighted variants can be solved by running a BFS from $s$ (or from $D_s$) in the underlying graph. Similarly, the decision problem in the weighted variants can be solved by running Dijkstra's shortest-path algorithm in the graph. However, the challenge is to do it efficiently, since the graph might have up to a quadratic number of edges. For unit-disk graphs, Cabello and Jejčič [5] presented an $O(n \log n)$ implementation of BFS, and subsequently Chan and Skrepetos [7] presented an alternative $O(n)$ implementation (after pre-sorting the points by their $x$- and $y$-coordinates). Moreover, Cabello and Jejčič [5] also described an $O(n^{1+\varepsilon})$ implementation of Dijkstra's algorithm for weighted unit-disk graphs, which was followed by a more efficient $O(n \log^2 n)$ implementation described by Wang and Xue [13]; see also [8].

The RSP problem in the context of unweighted unit-disk graphs was posed by Cabello and Jejčič [5], who observed that it can be solved conceptually easily in $O^*(n^{4/3})$ time, by running a binary search through the $O(n^2)$ inter-point distances (using an efficient distance selection algorithm). Recently, Wang and Zhao [14] managed to improve this bound, obtaining an algorithm that solves the problem in $O^*(n^{5/4})$ time. In the context of weighted unit-disk graphs, the situation is similar. The RSP problem can be solved easily in $O^*(n^{4/3})$ time, but Wang and Zhao [15] were able to obtain an improved $O^*(n^{5/4})$-time solution for that version too. [3]

As far as we know, both the decision and optimization problems have not been studied in the context of general disk graphs.

[3] The $O^*(n^{6/5})$ bound for the RSP problem in both unweighted and weighted unit disk graphs was already claimed in an unpublished manuscript [9], which appeared shortly after the first RSP paper of Wang and Zhao. However, this manuscript overlooks an issue that may arise when using an off-the-shelf decision problem, see the full version of this paper.

## 2 Preliminaries

In this section we provide necessary background on the serial parametric search technique of Ben Avraham et al. [4].

In its basic form, the technique is applicable when the threshold parameter $r^*$ is the distance between a pair of input points. There are $O(n^2)$ such distances, and the most naïve algorithm simply finds $r^*$ by running a binary search through them, guided by the decision procedure at each comparison. A basic improvement is to implement the binary search using an efficient procedure for distance selection, such as the one in [2] or [10], which runs in $O^*(n^{4/3})$ time. Up to an additional logarithmic factor, this dominates the cost of the whole procedure (assuming that the decision procedure is more efficient; as we show, this is indeed the case in all the RSP problems studied in this paper).

The technique of [4] is a combination of two subprocedures, referred to as the *interval shrinking* and the *bifurcation* procedures. The interval shrinking procedure receives an integer parameter $L \ll \binom{n}{2}$, and computes an interval $I \subset \mathbb{R}$ that contains $r^*$ and at most $L$ *critical values*, namely inter-point distances. As shown in [4], this can be done in $O^*(n^{4/3}/L^{1/3})$ expected time.

We then run the bifurcation procedure, which simulates the execution of the decision procedure at the (unknown) threshold $r^*$, as in the standard parametric search technique [12]. When the simulation reaches a comparison of $r^*$ with some concrete value $r$, we know the answer to the comparison when $r$ lies outside $I$. However, when $r \in I$ we bifurcate, following both possibilities $r^* > r$ and $r^* < r$ (the case $r^* = r$ will be handled too; see below). This produces a *bifurcation tree* $T$, which we expand until we either collect sufficiently many bifurcations, or until we reach a sufficiently large uniform depth of $T$. In either case we stop this simulation phase, resolve all collected comparisons by a binary search through them, using the (unsimulated) decision procedure to guide the search, and start a new bifurcation phase from the unique leaf of $T$ whose associated (shrunk) interval of critical values contains $r^*$. The binary search will also identify $r^*$ when it is one of the critical values through which it searches, and then terminate the entire procedure right away. In the worst case this will happen by the time when the entire decision procedure has been simulated.

We comment that this method is viable when the decision procedure is not known to have a parallel version of small depth, which is required in the standard parametric search technique. If such a parallel version were available, we could apply standard parametric search, and obtain a significantly faster algorithm. The RSP problem seems to be inherently sequential, as it seeks a path in a graph, and is indeed amenable to the technique of [4].

As shown in [4], the bifurcation procedure can be implemented to run in $O^*(L^{1/2}D(n))$ time, where $D(n)$ is the cost of the decision procedure. A suitable choice of $L$ yields an overall (randomized expected) running time $O^*(n^{6/5})$, for (suitable) decision procedures that run in nearly linear time, as do the decision procedures for all the variants of the RSP problem considered in this paper.

## 3 Reverse shortest paths for unweighted disk graphs

Here we are given a set $\mathcal{D}$ of $n$ disks in the plane, of arbitrary radii, each parameterized by its center $c_D$ and radius $\rho_D$. We consider the *intersection graph* $G^\times = (\mathcal{D}, E)$, where the edges of $G^\times$ are the intersecting pairs of disks. Formally, $E$ consists of all pairs $(D, D')$ for which $\|c_D - c_{D'}\| \leq \rho_D + \rho_{D'}$.

In the decision problem, we are given two designated disks $D_s$ and $D_t$, and an integer parameter $k$, and the goal is to determine whether $G^\times$ contains a path of at most $k$ edges from $D_s$ to $D_t$.

In the optimization problem we scale the radii of the disks (without changing their centers), either by an additive term or by a multiplicative factor, and seek the smallest scaling parameter that makes $G^\times$ have the desired $s$-$t$ path.

Towards the end of the section, we consider the special and important case of unit disk graphs, for which we obtain a better bound.

## 3.1 The decision procedure

We run BFS on $G^\times$ from $D_s$. Suppose that we have already discovered all disks at some level $i$ of the BFS. We expand the BFS to level $i+1$ as follows. We consider each disk $D$ at level $i$ in turn, and look for its nearest neighbor among the disks that have not yet been discovered. To do so, we maintain a dynamic additively-weighted Voronoi diagram $\mathrm{Vor}(\mathcal{U})$ of the set $\mathcal{U}$ of all the disks that the BFS has not yet reached, where the additive weight of a disk $D$ is $-\rho_D$. Initially, we are at level 0 of the BFS, which includes only $D_s$, and we set $\mathcal{U} = \mathcal{D} \setminus \{D_s\}$.

We search for the nearest neighbor $D'$ of $D$ in $\mathrm{Vor}(\mathcal{U})$. If the weighted distance between $D$ and $D'$ is at most $\rho_D$, that is, if $\|c_D - c_{D'}\| - \rho_{D'} \leq \rho_D$, we conclude that $\|c_D - c_{D'}\| \leq \rho_D + \rho_{D'}$, so we add $D'$ to level $i+1$, delete it from $\mathcal{U}$, and query the updated Voronoi diagram for the new nearest neighbor of $D$ again. We continue querying the updated $\mathrm{Vor}(\mathcal{U})$ with $D$ until the distance between $D$ and its nearest neighbor is larger than $\rho_D$. When this happens we replace $D$ by the next disk at level $i$ and repeat this process. When we finish processing in this manner all disks of level $i$, we have discovered all disks at level $i+1$, and we move on to level $i+1$.

Consider a sequence of queries to the Voronoi diagram with a disk $D$ at some level of the BFS. We can charge each of these queries but the last, to a new disk that we add, following this query, to the BFS tree. Therefore the running time of this decision procedure is dominated by the cost of $O(n)$ queries and $n$ deletions from $\mathrm{Vor}(\mathcal{U})$. The most efficient implementation of such a structure, with running time $O(n \log^4 n)$, is due to Liu [11]; see also the earlier study [8], with a worse polylogarithmic factor.

In summary, we have shown:

▶ **Theorem 1.** *Given $\mathcal{D}$, $D_s$, $D_t$, and $k$ as above, we can determine whether there exists a path of at most $k$ edges from $D_s$ to $D_t$ in the intersection graph $G^\times$ associated with $\mathcal{D}$, in $O(n \log^4 n)$ time.*

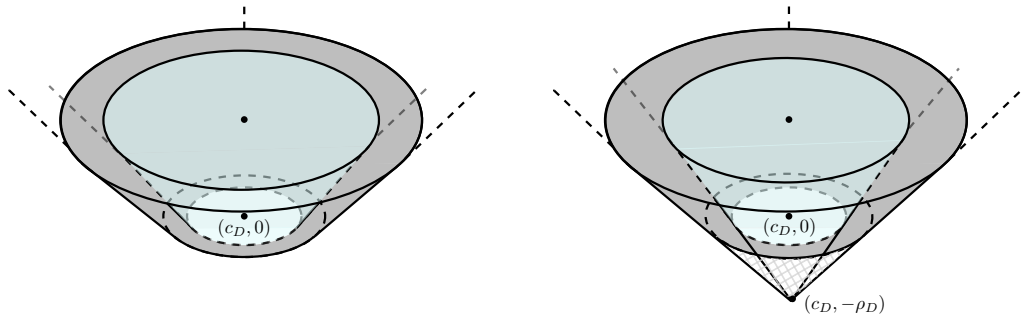
## 3.2 The optimization procedure

Each disk $D \in \mathcal{D}$ is assigned the radius $\rho_D + \alpha$, for some common additive parameter $\alpha$, and we want to find the minimum value $\alpha^*$ of $\alpha$ for which the intersection graph of the modified disks, now denoted by $G^\times_\alpha$, has the desired $s$-$t$ path. In principle $\alpha$ could also be negative, as long as no radius becomes negative, but for simplicity we only consider the case $\alpha > 0$.

We simulate the decision procedure at the unknown optimal value $\alpha^*$ by using a bifurcation procedure. Before starting the simulation, though, we perform an interval-shrinking step, as described in the introduction.

**Interval shrinking.** Recall that this step, as introduced in [4], receives an integer threshold parameter $L$ and produces an interval $I_0 \subset \mathbb{R}$ that contains $\alpha^*$ and at most $L$ other critical values, where a value $\alpha$ is critical if the outcome of a comparison changes as we go past $\alpha$. In the original formulation, the critical values were inter-point distances in the plane, and the resulting algorithm ran in $O^*(n^{4/3}/L^{1/3})$ randomized expected time.

Here the setup is different. The comparisons that the decision procedure performs are tests whether expressions of the form $\|c_D - c_{D'}\| - \rho_D - \rho_{D'} - 2\alpha$ are positive or negative. The critical values of the parameter $\alpha$ are thus of the form $\frac{1}{2}\left(\|c_D - c_{D'}\| - \rho_D - \rho_{D'}\right)$. The original mechanism of [4] is based on distance selection. We need a variant in which the basic step is to bound the number of these new critical values in a given interval $(\alpha_1, \alpha_2)$. We turn this problem into a range searching problem, where the disks of $\mathcal{D}$ serve as both data and query objects. Specifically, each disk $D$ is mapped to the point $(c_D, \rho_D)$ in $\mathbb{R}^3$, and also to the range $\sigma_D = \{D' \mid 2\alpha_1 \leq \|c_D - c_{D'}\| - \rho_D - \rho_{D'} \leq 2\alpha_2\}$, which is a conical shell in 3-space (see Figure 1(a)). Note that the ranges have three degrees of freedom, and that the problem is symmetric, so that ranges can be represented as points in $\mathbb{R}^3$ and points as ranges. In other words, we have a symmetric batched range searching problem in $\mathbb{R}^3$, involving $n$ points and $n$ semi-algebraic ranges. Using standard, cutting-based decomposition techniques in $\mathbb{R}^3$, such as in [1, 3], we can implement the range searching step to run in randomized expected time $O^*(n^{3/2})$. Combining this with parametric search, as in the standard distance selection procedure [2], we can implement the distance selection procedure to also run in randomized expected time $O^*(n^{3/2})$.




(a) (b)

**Figure 1** The range $\sigma_D$ (in grey) when scaling by an additive term (a) and by a multiplicative factor (b). The inner and outer radii of the annulus centered at $(c_D, 0)$ are $\rho_D + 2\alpha_1$ and $\rho_D + 2\alpha_2$, respectively, in (a) and $\rho_D\lambda_1$ and $\rho_D\lambda_2$, respectively, in (b).

Extending the machinery in [4], we can convert this distance selection technique to an interval-shrinking procedure that receives a parameter $L \ll \binom{n}{2}$ and yields an interval $I = (\alpha_1, \alpha_2)$ that contains the optimum value $\alpha^*$ and at most $L$ critical values. A suitable modification of the analysis in [4] shows that the algorithm runs in $O^*(n^{3/2}/L^{1/2})$ randomized expected time.

**Bifurcation.** We now present the basic bifurcation procedure. This procedure, sometimes with a few enhancements and modifications, is used in all our algorithms for the various variants of the RSP problem. We refer the reader to [4] where a similar procedure has been used.

Our simulation proceeds in *phases*, where in each phase we construct a bifurcation tree $T$ that represents some portion of the execution of the BFS, simultaneously for all values of $\alpha$ in some interval $I$. Initially $I = I_0$, but it will keep on shrinking as the simulation proceeds. Each node $b$ of $T$ is associated with an interval $I^b = (\alpha_1, \alpha_2) \subseteq I$, such that, up to the state of simulation represented by $b$, the BFS proceeds in an identical manner for all values $\alpha \in I^b$. We continue to simulate the BFS at $b$. At each comparison of (the unknown) $\alpha^*$ with some concrete value $\alpha$, we either resolve the comparison in a unique manner when $\alpha \notin I^b$, or

else bifurcate, meaning that we create two children $u$ and $v$ of $b$, assign $I^u := (\alpha_1, \alpha)$ and $I^v := (\alpha, \alpha_2)$, and continue to expand $T$ at $u$ and at $v$, at each of which we know the outcome of the above procedure (the possibility that $\alpha^* = \alpha$ will be tested later).

For each node $b$, let $y_b$ denote the amount of work performed at $b$ so far by the simulation (including comparisons that were fully resolved), and let $y_b^-$ denote the sum of the quantities $y_a$ over all proper ancestors $a$ of $b$, *excluding the root*. We refer to the cost at the root as the *initialization cost* of the tree, and denote it as $C_0(T)$.

We stop the expansion of $T$ at a node $b$ when $y_b^- + y_b = Y$, where $Y$ is a threshold parameter that will be determined later. That is, we stop the simulation at node $b$ as soon as $y_b^-$ plus the work done so far at $b$ becomes $Y$. We refer to such nodes $b$ as *incomplete leaves* of $T$. We then continue the expansion of $T$ at other nodes.

A subtle issue, that we will address later in more detail, is that when we back up from an incomplete node $b$ to explore other branches of $T$, we need to restore the state of execution at the suitable ancestors of $b$. See below for details.

We stop the entire construction of $T$ as soon as one of the following two conditions occurs:

- **(i)** We collect $X$ bifurcations, for another threshold parameter $X$ that will be determined later.
- **(ii)** All the leaves of $T$ are incomplete.

When this happens, we take all the (at most $X$) critical values of the bifurcations at the inner nodes of $T$, and run a binary search through them, using the unsimulated decision procedure to guide the search. This takes $O(D(n) \log n)$ time, and yields the leaf $w$ whose range $I^w$ contains $r^*$. It is also possible that the binary search will detect that one of the critical values it searches through is $r^*$ itself. In this case the entire procedure is terminated, and $r^*$ is output. Otherwise, this ends a phase of the simulation. We start a new phase (if the simulation has not already ended) with $w$ as the root of the tree, and with $I^w$ as the critical interval. Note that this will cause the work already done at $w$ to be repeated, and it is possible that the cost of this work is much larger than $Y$. However, by charging the incomplete work at $w$ to the initialization cost $C_0(T')$ of the next tree $T'$, we at most double this cost, so this will not affect the overall asymptotic bound on the performance of the procedure; see below for details.

**Restoring the execution state.** The issue of restoring the state at nodes we back up to, as mentioned earlier, is more acute in our specific application, because part of this state includes the Voronoi diagram that our procedure maintains dynamically. Although there are persistence-based techniques that can efficiently maintain all versions of the diagram, they are fairly involved, and we opt not to use them. Instead, we use the following simple approach, which also takes care of all aspects of restoring the state.

Specifically, we expand the bifurcation tree in a depth-first manner, so that at each node we first recursively construct the subtree of its left child and only then the subtree of its right child. We thus first expand the leftmost path of $T$, and slowly proceed to the right, backing from a node to its parent, and then proceed to the right child, or further up to the grandparent. When we back up from a node $w$ to its parent $v$, we simply undo the operations performed at $w$, including the updates that were done to the diagram, in reverse order, replacing each deletion of a disk by the corresponding reinsertion. This requires us to maintain a log of the updates performed at each node, as well as a log of the other operations, but is otherwise a reasonably simple procedure, which does not affect the asymptotic running time and storage bounds.

## 3.3 Analysis

By construction, a single phase produces a tree $T$ that has at most $X$ binary nodes, and the cost of producing each path of $T$ is at most $Y$, ignoring the initialization cost $C_0(T)$. This is easily seen to imply that the cost of generating $T$ is $O(XY + C_0(T))$. Indeed, the cost at each node of $T$, other than the root, is certainly at most $Y$, and there are $O(X)$ such nodes The cost of the subsequent binary search through the critical values is $O(D(n) \log n)$. Hence the overall cost of a single phase is $O(XY + C_0(T) + D(n) \log n)$. The number of phases is estimated as follows. If the phase terminates because of Condition (i), it has discovered $X$ critical values among those in the current critical interval, which are outside the new critical interval $I^w$. Hence the number of such phases is at most $L/X$.[4]

A phase that terminates because of Condition (ii) consumes $Y$ of the total cost of the decision procedure: when we pass to the next phase we follow a single path of the current $T$, but all paths use at least $Y$ of the cost, excluding the work done at the root. Note that, by construction, the part of the simulation performed along a path of the tree in one phase, excluding the work performed at the root, is disjoint from the part performed along a path of the tree in a different phase. This is easily seen to imply that the number of phases of type (ii) is at most $D(n)/Y$.

The sum $\sum_T C_0(T)$ of the initialization costs of all the trees is at most $D(n)$, because these initializations perform pairwise disjoint portions of the decision procedure. (Observe that execution at a root is always completed, no matter how expensive it is.)

We set our parameters so that $\frac{L}{X} = \frac{D(n)}{Y}$ and $XY = D(n) \log n$. That is, we choose $X = L^{1/2} \log^{1/2} n$ and $Y = D(n) \log^{1/2} n / L^{1/2}$, and obtain that the overall cost of the simulation is $O(L^{1/2} D(n) \log^{1/2} n)$ (this clearly also subsumes the cost $\sum_T C_0(T)$).

In total, the cost of the optimization procedure is

$$O^*\left(\frac{n^{3/2}}{L^{1/2}} + L^{1/2} D(n)\right) = O^*\left(\frac{n^{3/2}}{L^{1/2}} + L^{1/2} n\right).$$

We balance these two terms by choosing $L = n^{1/2}$, and obtain a total cost of $O^*(n^{5/4})$. This result is part of the summary in Theorem 2 (which also includes other variants of the problem), given below.

## 3.4 Other variants and unit disks

**Scaling by a multiplicative factor.** Consider first the case of intersection graphs where the radii are scaled by a common multiplicative factor $\lambda > 0$. In this case the critical value induced by a pair of disks $D$, $D'$ satisfies $\|c_D - c_{D'}\| - \lambda\rho_D - \lambda\rho_{D'} = 0$, or $\lambda = \dfrac{\|c_D - c_{D'}\|}{\rho_D + \rho_{D'}}$. As above, the algorithm requires a procedure that computes the number of critical values in an interval $(\lambda_1, \lambda_2)$. So we convert this task to batched range searching in three dimensions, where the ranges are now

$$\sigma_D = \{D' \mid \lambda_1(\rho_D + \rho_{D'}) \le \|c_D - c_{D'}\| \le \lambda_2(\rho_D + \rho_{D'})\}.$$

[4] This is a rather weak aspect of the analysis. For the bound $L/X$ to materialize, the $X$ critical values of each phase must form a prefix or a suffix of the sequence of critical values in the current interval $I$. In general, when these critical values are more uniformly spread within $I$, the number of such phases should be much smaller. It is a challenging open problem to turn this intuition into an improved procedure, if possible.

These too are conical shells, but here the bounding cones have a common apex and different opening angles (see Figure 1(b)). Other than this new type of ranges, the preceding machinery proceeds verbatim. The interval shrinking runs in $O^*(n^{3/2}/L^{1/2})$ expected time and the bifurcation procedure runs in $O^*(L^{1/2}n)$. With a proper choice of $L$, the overall cost is $O^*(n^{5/4})$ expected time.

**Proximity graphs.** Next, consider the case of proximity graphs. In this case, we assume that the disks of $\mathcal{D}$ are pairwise disjoint, and we are given an additional parameter $r > 0$. (For $r = 0$ we get the intersection graph $G^\times$.) The set of edges of the proximity graph $G_r$ consists of all pairs of disks $(D, D')$ for which $\mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'} \leq r$. In the RSP problem for proximity graphs, we seek the smallest value of $r$ for which $G_r$ has the desired $s$-$t$ path. It is easy to see that this problem can be reduced to the (additive version of the) corresponding problem for intersection graphs, by simply adding $r/2$ to the radius of each of the disks. In the optimization procedure, the critical values are of the form $\|c_D - c_{D'}\| - \rho_D - \rho_{D'}$. Thus, except for the factor $\frac{1}{2}$ used earlier, the procedure is essentially identical to the earlier one.

**Unit disks.** Finally, consider the special case of unit disks, i.e., where all the radii are equal, and consider the intersection graph of the disks. In this case, we get a better bound, since the critical values are merely (one half of the) inter-point distances, and hence the interval shrinking step can be performed in $O^*(n^{4/3}/L^{1/3})$ randomized expected time, as in [4]. Modifying the expression for the overall running time accordingly, we get $O^*\left(\frac{n^{4/3}}{L^{1/3}} + L^{1/2}n\right)$ randomized expected time. We now balance these two terms by choosing $L = n^{2/5}$, and obtain a total cost of $O^*(n^{6/5})$. In summary, we have shown:

▶ **Theorem 2.** *The reverse shortest path problem for unweighted intersection or proximity graphs of arbitrary disks in the plane can be solved in $O^*(n^{5/4})$ randomized expected time. This bound applies to all the variants of the problem listed above. In the case of unit disks, where all radii are equal, the problem can be solved in $O^*(n^{6/5})$ randomized expected time.*

**Remark.** In the case of unit disks, the decision procedure itself can be implemented to run faster than $O(n \log^4 n)$. Chan and Skrepetos [7] even present an algorithm that runs in linear time (after a preliminary sorting step); see also [5]. However, the critical values produced by their procedure are not all inter-point distances, which affects the running time of the optimization procedure. To avoid the use of complex dynamic data structures, we modify the algorithm of Chan and Skrepetos, so that it can be combined with the optimization procedure, see the full version of this paper.

## 4 Reverse shortest paths for weighted disk graphs

Recall that, for a set $\mathcal{D}$ of $n$ disks in the plane and an additional parameter $r \geq 0$, the proximity graph $G_r$ of $\mathcal{D}$ has an edge between every pair of disks such that

$$\mathsf{dist}(D, D') := \|c_D - c_{D'}\| - \rho_D - \rho_{D'} \leq r. \tag{1}$$

The intersection graph $G^\times$ of $\mathcal{D}$ is the proximity graph of $\mathcal{D}$ for $r = 0$, except that in proximity graphs we usually assume that the disks are pairwise disjoint.

We next assign weights to the edges. There are two natural choices for these weights. One is to define the weight of an edge $(D, D')$ as the distance $\|c_D - c_{D'}\|$ between the centers. The other is to define the weight of $(D, D')$ to be $\mathsf{dist}(D, D')$ if $\mathsf{dist}(D, D') \geq 0$ and 0 otherwise (that is 0 if the disks intersect). Note that for intersection graphs only the first choice gives a reasonable weight function.

The case of weighted unit disk graphs is a special case of the problem for intersection graphs $G^{\times}$ with weights equal to the distances between the centers. In this case we have $\|c_D - c_{D'}\| = \mathsf{dist}(D, D') + 2$.

## 4.1 The decision procedure

We are given two disks $D_s$, $D_t$ of $\mathcal{D}$, and the proximity graph $G_r$ of $\mathcal{D}$ for some $r \geq 0$, weighted by one of the weight functions above. We want to compute the (length of the) shortest path $\pi(D_s, D_t)$ in $G$ from $D_s$ to $D_t$. We solve this by a clever implementation of Dijkstra's algorithm. The required sophistication of the implementation depends on the type of the weight function we use. We start with the simpler case in which the weight of $(D, D')$ is $\mathsf{dist}(D, D')$ (or 0 if the disks intersect). Specifically, the same function that defines the graph via the threshold $r$ (Equation (1)) is also used to define the weights.

The high-level approach is similar to that proposed by Cabello and Jejčič [5], although their original algorithm was given only for intersection graphs of unit disks. We briefly recall this technique, adapted to our context.

We maintain a decomposition of $\mathcal{D}$ into three disjoint subsets $\mathcal{R}$, $\mathcal{K}$ and $\mathcal{U}$, where $\mathcal{R} \cup \mathcal{K}$ is the set of disks $D$ for which we already have the correct distance label $\delta(D)$ (the length of the shortest path from $D_s$), and $\mathcal{U}$ is the remainder of $\mathcal{D}$, consisting of disks whose distance labels have not yet been determined. $\mathcal{R}$ (resp., $\mathcal{K}$) is the set of *active* (resp., *dead*) disks of $\mathcal{R} \cup \mathcal{K}$, meaning that the disks of $\mathcal{R}$ still have outgoing edges in $G_r$ to disks of $\mathcal{U}$, while disks of $\mathcal{K}$ have no such edges. Initially, $\mathcal{R} = \{D_s\}$, $\mathcal{K} = \emptyset$, and $\mathcal{U} = \mathcal{D} \setminus \{D_s\}$.

A single step of our implementation of Dijkstra's algorithm, a so-called *Dijkstra step*, picks the closest pair $(D, D')$ in $\mathcal{R} \times \mathcal{U}$, where the modified distance between $D$ and $D'$ is defined as

$$\mathsf{d}(D, D') = \delta(D) + \mathsf{dist}(D, D'). \tag{2}$$

Then we need to verify that $(D, D')$ is indeed an edge of $G_r$. If this is not the case, i.e., $\mathsf{dist}(D, D') > r$, we move $D$ from $\mathcal{R}$ to $\mathcal{K}$, and never process $D$ again. This action is justified by the following variant of [5, Lemma 6]:

▶ **Lemma 3.** *Let $D$ and $D'$ be as above, and assume that $\mathsf{dist}(D, D') > r$. Then $\mathsf{dist}(D, D'') > r$ for every disk $D'' \in \mathcal{U}$.*

**Proof.** By assumption, and since $(D, D')$ is the closest pair in $\mathcal{R} \times \mathcal{U}$, we have, for each $D'' \in \mathcal{U}$, $\delta(D) + r < \delta(D) + \mathsf{dist}(D, D') \leq \delta(D) + \mathsf{dist}(D, D'')$, so $\mathsf{dist}(D, D'') > r$, as asserted. ◀

Note that Lemma 3 also applies to weighted unit disk graphs (with weight $\|c_D - c_{D'}\|$ for edge $(D, D')$.)

Assume then that $\mathsf{dist}(D, D') \leq r$. We move $D'$ from $\mathcal{U}$ to $\mathcal{R}$, assign to it the distance label $\delta(D') = \delta(D) + \mathsf{dist}(D, D')$, and set $\mathsf{prev}(D') = D$, namely $D$ is the disk preceding $D'$ along the shortest path to $D'$. We then repeat the entire step with the new sets $\mathcal{R}$ and $\mathcal{U}$, and continue until $\mathcal{R}$ empties out. Upon termination, $\mathcal{U}$ is the set of disks that are unreachable from $D_s$ in $G_r$, and $\mathcal{K}$ is the set of reachable disks, each with its correct distance label and its predecessor.

The correctness of this procedure is argued exactly as in [5], even though the distance function is different. An efficient implementation is obtained by applying the efficient dynamic bichromatic closest-pair data structure of Kaplan et al. [8] and of Liu [11] to $\mathcal{R} \times \mathcal{U}$, under the distance function given in (2) and under deletions from $\mathcal{U}$ and insertions into and deletions from $\mathcal{R}$. The technique requires the following two properties.

- **(i)** The Voronoi diagram $\mathrm{Vor}_1(\mathcal{U})$ of $\mathcal{U}$, under the distance function $\mathsf{d}_1(q, D') = \|q - c_{D'}\| - \rho_{D'}$, for $D' \in \mathcal{U}$, where $c_{D'}$ and $\rho_{D'}$ are the respective center and radius of $D'$, has linear complexity.
- **(ii)** The Voronoi diagram $\mathrm{Vor}_2(\mathcal{R})$ of $\mathcal{R}$, under the distance function $\mathsf{d}_2(q, D) = \delta(D) + \|q - c_D\| - \rho_D$, for $D \in \mathcal{R}$, also has linear complexity.

Both properties indeed hold, as each diagram is an additively-weighted Voronoi diagram of the set of centers of the disks of $\mathcal{U}$ or of $\mathcal{R}$. Both diagrams need to be maintained dynamically, and the techniques of [8, 11] do that, with a polylogarithmic cost for each update and query operation. In the more efficient implementation of [11], the amortized cost of an update is $O(\log^4 n)$.

In summary, we have shown:

▶ **Theorem 4.** *The shortest-path tree from some starting disk in the proximity graph of $n$ disks (of arbitrary radii) in the plane, under the weights of inter-disk distances, can be constructed in $O(n \log^4 n)$ time.*

Next we address the more challenging weighted case when the edge weights are the distances between the centers.

**The edge weights are the distances between the centers.** In this case we need to maintain a dynamic bichromatic closest pair structure under the distance $\lambda(D, D') = \delta(D) + \|c_D - c_{D'}\|$, for $D \in \mathcal{R}$ and $D' \in \mathcal{U}$. But then Lemma 3 does not hold anymore, because it is then possible that the closest pair $(D, D') \in \mathcal{R} \times \mathcal{U}$ satisfies $\mathsf{dist}(D, D') > r$ but there could be other pairs $(D, D'')$ with $\mathsf{dist}(D, D'') \leq r$, so $D$ cannot be removed from $\mathcal{R}$ yet.

To overcome this difficulty we obtain a *bichromatic closest pair* (BCP for short) data structure by applying a black-box reduction of Chan [6] to two novel nearest-neighbor data structures that we now describe.

Our new data structures find for $D \in \mathcal{R}$ a nearest neighbor $D' \in \mathcal{U}$ according to the distance function $\lambda$ but only among disks $D'$ such that $dist(D, D') \leq r$, and similarly for $D \in \mathcal{U}$.

We then run the Dijkstra steps, using the BCP data structure, as long as there are still neighbors in $\mathcal{R} \times \mathcal{U}$. Upon termination, $\mathcal{U}$ is the set of disks that are unreachable from $D_s$ in $G_r$, and $\mathcal{R}$ is the set of reachable disks, each with its correct distance label and its predecessor.

Our nearest-neighbor data structure consists of two balanced search trees $T_{\mathcal{U}}$ and $T_{\mathcal{R}}$. The tree $T_{\mathcal{U}}$ ($T_{\mathcal{R}}$ is handled in a similar manner; see below) stores the disks $D' \in \mathcal{U}$ at its leaves, sorted in increasing order of the values $\rho_{D'}$. For each node $v$ of $T_{\mathcal{U}}$, we maintain an additively-weighted Voronoi diagram $\mathrm{Vor}_1(\mathcal{U}_v)$ on the set $\mathcal{U}_v$ of the disks of $\mathcal{U}$ stored at the leaves of the subtree rooted at $v$, where the weight of a disk $D'$ is $-\rho_{D'}$. We also maintain a second standard (unweighted) Voronoi diagram $\mathrm{Vor}_2(\mathcal{U}_v)$ for $\mathcal{U}_v$.

Querying $T_{\mathcal{U}}$ with a disk $D \in \mathcal{R}$ is performed as follows. We find the leftmost leaf $w_0$ of $T_{\mathcal{U}}$ whose disk $D'_0$ satisfies $\mathsf{dist}(D, D'_0) = \|c_D - c_{D'_0}\| - \rho_D - \rho_{D'_0} \leq r$. To find $w_0$, we search in $T_{\mathcal{U}}$ starting at the root. At each node $u$ that we reach, with a left child $v$ and a right child

$w$, we search with $D$ in $\mathrm{Vor}_1(\mathcal{U}_v)$, and obtain its nearest neighbor $D'$ in the corresponding subset of disks. If $\|c_D - c_{D'}\| - \rho_{D'} \leq \rho_D + r$, or, equivalently, $\mathsf{dist}(D, D') \leq r$, then we continue the search at the left child $v$. Otherwise we continue the search with the right child $w$. At the root we first search in $\mathrm{Vor}_1(\mathcal{U}_{\mathrm{root}} = \mathcal{U})$. If the nearest neighbor $D'$ satisfies $\|c_D - c_{D'}\| - \rho_{D'} > \rho_D + r$, that is, $\mathsf{dist}(D, D') > r$, we conclude that $D$ has no neighbor in $\mathcal{U}$.

Let $w_0$ be the leaf that the search reaches. Note that if a disk $D'$ is stored at a leaf to the left of $w_0$ then, by construction, $\mathsf{dist}(D, D') > r$. That is, only disks stored to the right of $w_0$, including $w_0$, need to be considered.

The search for $w_0$ provides us with a representation of the set $\mathcal{U}^+_{w_0}$ of the disks to the right of $w_0$ as the union of $O(\log n)$ pairwise disjoint canonical sets, each being the set of disks stored at the root of some subtree of $T_{\mathcal{U}}$. We query with $D$ in each of the $O(\log n)$ diagrams $\mathrm{Vor}_2(\mathcal{U}_v)$ that correspond to these subtrees, and return the disk that is nearest to $D$, i.e., with the minimum distance between their centers, among all the resulting nearest neighbors. The correctness of this procedure is a consequence of the following lemma.

▶ **Lemma 5.** *In the above procedure, the output disk $D'$ satisfies $\mathsf{dist}(D, D') \leq r$.*

**Proof.** Let $D'_0$ be the disk at the leaf $w_0$. By construction, we have

$$\rho_{D'_0} \leq \rho_{D'}. \tag{3}$$

By construction, $D'$ is the disk in $\mathcal{U}^+_{w_0}$ nearest to $D$ in the inter-center distance. That is,

$$\|c_D - c_{D'}\| \leq \|c_D - c_{D'_0}\|. \tag{4}$$

Assume by contradiction that $\mathsf{dist}(D, D') > r$. Then, by construction,

$$\|c_D - c_{D'_0}\| - \rho_{D'_0} - \rho_D = \mathsf{dist}(D, D'_0) \leq r < \mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_{D'} - \rho_D.$$

That is,

$$\|c_D - c_{D'_0}\| - \rho_{D'_0} < \|c_D - c_{D'}\| - \rho_{D'}. \tag{5}$$

Adding (3) and (5), we get a contradiction to (4). This establishes the lemma. ◀

The tree $T_{\mathcal{R}}$ is defined in an analogous manner for the disks of $\mathcal{R}$, except that (a) the leaves are sorted in increasing order of $\delta(D) + \rho_D$, and (b) the Voronoi diagram $\mathrm{Vor}_2(\mathcal{R}_v)$ at a node $v$ of $T_{\mathcal{R}}$, where $\mathcal{R}_v$ is the set of disks stored at the root of the subtree rooted at $v$, is the additively-weighted diagram on $\mathcal{R}_v$, where the additive weight of a disk $D$ is $\delta(D)$.

The first kind of Voronoi diagrams $\mathrm{Vor}_1(\mathcal{R}_v)$ are defined exactly as in the case of $T_{\mathcal{U}}$, with the additive weight being $-\rho_D$.

Here too, when we query $T_{\mathcal{R}}$ with a disk $D'$ of $\mathcal{U}$, we search in the tree to find the leftmost leaf $w_0$ of $T_{\mathcal{R}}$ whose disk $D_0$ satisfies $\mathsf{dist}(D_0, D') = \|c_{D_0} - c_{D'}\| - \rho_{D_0} - \rho_{D'} \leq r$. This is performed as follows. We first search for the leftmost leaf $w_0$ whose associated disk $D_0$ satisfies $\mathsf{dist}(D_0, D') \leq r$, using the same technique as in the previous search in $T_{\mathcal{U}}$, in which we query various Voronoi diagrams $\mathrm{Vor}_1(\mathcal{R}_v)$ along the search path to $w_0$. We then obtain the set $\mathcal{R}^+_{w_0}$ of all disks stored at the leaves to the right of $w_0$, including $w_0$, as the disjoint union of $O(\log n)$ subtrees. We query each of the diagrams $\mathrm{Vor}_2(\mathcal{R}_v)$ associated with these subtrees, and return the disk $D$ that is the nearest neighbor to $D'$ over all these diagrams.

The correctness of this procedure is a consequence of the following "sister" lemma to Lemma 5.

▶ **Lemma 6.** *In the above procedure, the output disk $D$ satisfies* $\mathsf{dist}(D, D') \leq r$.

**Proof.** Let $D_0$ be the disk at the leaf $w_0$. By construction, we have

$$\delta(D_0) + \rho_{D_0} \leq \delta(D) + \rho_D. \tag{6}$$

By construction, $D$ is the disk in $\mathcal{U}^+_{w_0}$ nearest to $D'$ in the inter-center distance with the additive weight $\delta$. That is,

$$\delta(D) + \|c_D - c_{D'}\| \leq \delta(D_0) + \|c_{D_0} - c_{D'}\|. \tag{7}$$

Assume by contradiction that $\mathsf{dist}(D, D') > r$. Then, by construction,

$$\|c_{D_0} - c_{D'}\| - \rho_{D_0} - \rho_{D'} = \mathsf{dist}(D_0, D') \leq r < \mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'}.$$

That is,

$$\|c_{D_0} - c_{D'}\| - \rho_{D_0} < \|c_D - c_{D'}\| - \rho_D. \tag{8}$$

Adding (6) and (8), we get

$$\delta(D_0) + \|c_{D_0} - c_{D'}\| < \delta(D) + \|c_D - c_{D'}\|,$$

which is a contradiction to (7). This establishes the lemma. ◀

We make these nearest-neighbor data structures dynamic by maintaining the various Voronoi diagrams $\mathrm{Vor}_1(\mathcal{U}_v)$, $\mathrm{Vor}_2(\mathcal{U}_v)$, $\mathrm{Vor}_1(\mathcal{R}_v)$, $\mathrm{Vor}_2(\mathcal{R}_v)$, dynamically, using the technique of [8, 11], in which the update time of each diagram is $O(\log^4 n)$. An update of either of the trees $T_{\mathcal{U}}$, $T_{\mathcal{R}}$ requires updating $O(\log n)$ data structures along the path to the leaf containing the inserted or deleted element, and therefore takes $O(\log^5 n)$ time. The transformation of Chan [6] from the two nearest-neighbor structures to a BCP data structure incurs an additional logarithmic overhead. Overall we get

▶ **Theorem 7.** *The shortest-path tree from some starting disk in the proximity graph of $n$ disks (of arbitrary radii) in the plane, under the weights of inter-center distances, can be constructed in $O(n \log^6 n)$ time.*

**Remark.** As in the unweighted version, in the case of unit disks, the decision procedure itself can be implemented to run faster than $O(n \log^6 n)$. For example, Wang and Xue [13] present an algorithm that runs in $O(n \log^2 n)$ time, but this algorithm is not suitable for our optimization procedure, since its comparisons generate critical values that are determined by more than two disks. We thus replace its "problematic" components by new ones, to obtain a (somewhat simpler) algorithm that is suitable for the optimization procedure, see the full version of this paper.

## 4.2 The optimization procedure

The most natural reverse shortest path question is to find the smallest value of $r$ such that the length of the shortest path in $G_r$ from $D_s$ to $D_t$ is smaller than some given real parameter $w$.

This optimization procedure is implemented as in the unweighted case, using the same combination of the interval shrinking and bifurcation procedures. The critical values are the same as in the case for the unweighted proximity graph.

In summary, we have shown:

▶ **Theorem 8.** *The reverse shortest path problem for weighted intersection or proximity graphs of arbitrary disks in the plane can be solved in $O^*(n^{5/4})$ randomized expected time. This bound applies to all the variants of the problem listed above. In the case of unit disks, where all radii are equal, the problem can be solved in $O^*(n^{6/5})$ randomized expected time.*

**Remark.** The machinery developed in this section seems to be more broadly applicable to other optimization questions that involve shortest paths in weighted proximity or intersection graphs, of disks or of more general geometric objects. Simple extensions could involve different definitions of proximity or the use of other weight functions. Other extensions could involve different problems, such as computing all-pairs shortest paths, computing shortest paths with negative edge weights, computing the diameter of the graph, etc.

## References


1 P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.

2 P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.

3 P. K. Agarwal, M. J. Katz, and M. Sharir. On reverse shortest paths in geometric proximity graphs. In *33rd Int. Sympos. on Algorithms and Computation (ISAAC)*, pages 42:1–42:19, 2022.

4 R. Ben Avraham, O. Filtser, H. Kaplan, M. J. Katz, and M. Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Trans. Algorithms*, 11, 2015, Art. 29.

5 S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom. Theory Appls.*, 48:360–367, 2015.

6 T. M. Chan. Dynamic generalized closest pair: Revisiting Eppstein's technique. In *Proc. 3rd Sympos. Simplicity in Algorithms (SOSA)*, pages 33–37, 2020.

7 T. M. Chan and D. Skrepetos. All-pairs shortest paths in unit disk graphs in slightly subquadratic time. In *Proc. 27th Int. Sympos on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

8 H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete Comput. Geom.*, 64:838–904, 2020.

9 M. J. Katz and M. Sharir. Efficient algorithms for optimization problems involving semi-algebraic range searching. in `arXiv:2111.02052`.

10 M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26:1384–1408, 1997.

11 C.-H. Liu. Nearly optimal planar $k$ nearest neighbors queries under general distance functions. *SIAM J. Comput.*, 51(3):723–765, 2022.

12 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. of the ACM*, 30:852–865, 1983.

13 H. Wang and J. Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete Comput. Geom.*, 64:1141–1166, 2020.

14 H. Wang and Y. Zhao. Reverse shortest path problem for unit-disk graphs. In *Proc. 17th Algorithms and Data Structures Sympos. (WADS)*, pages 655–668, 2021.

15 H. Wang and Y. Zhao. Reverse shortest path problem in weighted unit-disk graphs. In *Proc. 16th Int. Conf. and Workshops on Algorithms and Computation (WALCOM)*, pages 135–146, 2022.

# On Fully Dynamic Strongly Connected Components

**Adam Karczmarz**
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

**Marcin Smulewicz**
University of Warsaw, Poland

## Abstract

We consider maintaining strongly connected components (SCCs) of a directed graph subject to edge insertions and deletions. For this problem, we show a randomized algebraic data structure with conditionally tight $O(n^{1.529})$ worst-case update time. The only previously described subquadratic update bound for this problem [Karczmarz, Mukherjee, and Sankowski, STOC'22] holds exclusively in the amortized sense.

For the less general dynamic strong connectivity problem, where one is only interested in maintaining whether the graph is strongly connected, we give an efficient deterministic black-box reduction to (arbitrary-pair) dynamic reachability. Consequently, for dynamic strong connectivity we match the best-known $O(n^{1.407})$ worst-case upper bound for dynamic reachability [van den Brand, Nanongkai, and Saranurak FOCS'19]. This is also conditionally optimal and improves upon the previous $O(n^{1.529})$ bound. Our reduction also yields the first fully dynamic algorithms for maintaining the minimum strong connectivity augmentation of a digraph.

**2012 ACM Subject Classification** Theory of computation → Dynamic graph algorithms

**Keywords and phrases** dynamic strongly connected components, dynamic strong connectivity, dynamic reachability

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.68

**Funding** *Adam Karczmarz*: Partially supported by the ERC CoG grant TUgbOAT no 772346 and the National Science Centre (NCN) grant no. 2022/47/D/ST6/02184.

## 1 Introduction

Two vertices of a directed graph $G = (V, E)$ are *strongly connected* if they can reach each other via a path in $G$. Pairwise strong connectivity is an equivalence relation and the *strongly connected components* of $G$ are the equivalence classes of that relation. The graph $G$ is called *strongly connected* if it has a single strongly connected component, or, equivalently, the transitive closure of $G$ is a complete digraph. Testing strong connectivity and computing strongly connected components (SCCs) are among the most fundamental algorithmic problems on digraphs. Tarjan [24] famously showed a DFS-based linear-time algorithm for finding SCCs, which now, along with later alternatives [11, 23], is often taught in basic algorithms courses and appears in textbooks [9].

We study maintaining strong connectivity and strongly connected components in dynamic setting. A dynamic graph data structure is called *incremental* if it can handle edge insertions only, *decremental* if it can handle edge deletions only, and *fully dynamic* if it can handle both. When designing dynamic graph data structures, one is typically interested in optimizing both the (amortized or worst-case) update time of the data structure, and the query time. In the partially dynamic settings (i.e., incremental or decremental) one usually optimizes the total update time, i.e., the time needed to process the entire sequence of updates. In the following, let $n = |V|$ and $m = |E|$.


In this paper, we focus on the fully dynamic setting with single-edge updates. We are interested in maintaining the information about the *global* structure of SCCs of $G$, as opposed to supporting pairwise strong-connectivity queries. The information of interest may be either:

**(a)** a single bit indicating whether $G$ is strongly connected (let us call this variant *SC*),
**(b)** the number of SCCs of $G$ (*#SCC*),
**(c)** a representation of the SCCs allowing very efficient access to the individual SCCs, that is, computing the size in $O(\text{polylog}\, n)$ time, or reporting the elements in $O(\text{polylog}\, n)$ time per element (*SCCs*).

In particular, (c) can be achieved by maintaining an explicit $\Theta(n)$-sized partition of $V$ into SCCs, which also allows for $O(1)$-time pairwise strong connectivity queries. On the other hand, using pairwise strong connectivity queries cannot easily provide any of the considered global information. In the following, for simplicity, we assume the desired information is explicitly recomputed after each update so that there is no need to consider query time. Thus, our sole goal is to optimize the *worst-case* update time.

**Lower bounds.** Note that if one is required to maintain the SCCs of a fully dynamic graph explicitly, then a single update can cause quite dramatic $\Omega(n)$-sized change in the set of SCCs (consider a directed cycle). Abboud and Vassilevska Williams [1] showed that even for maintaining a single-bit information whether $G$ has more than two SCCs (the *SC2* problem), a data structure with $O(n^{1-\epsilon})$ amortized time[1] is unlikely, as it would break the Orthogonal Vectors conjecture, which is implied by SETH [15, 27]. The SETH-hardness of [1] does not seem to extend to *SC* though, which suggests that #SCC (or even SC2) might be a computationally harder problem.

The more recent OMv conjecture [14] implies that one cannot achieve $O(n^{1-\epsilon})$ update time even for SC. This suggests that for sparse graphs with $m = \widetilde{O}(n)$ the trivial recompute-from-scratch approach might be the best possible approach for SC and SCCs.

van den Brand, Nanongkai, and Saranurak [26] developed a few extensions of the OMv conjecture and proved, based on those, that fully dynamic $s,t$-reachability (where $s,t \in V$ are fixed) currently requires $\Omega(n^{1.406})$ amortized update time, and fully dynamic single-source reachability (with only the source $s$ fixed) currently requires $\Omega(n^{1.528})$ time per update.[2] This means that the state-of-the-art data structures [26, 20] for the respective variants are near-optimal. By known reductions (see, e.g., [1]), one obtains that SC requires $\Omega(n^{1.406})$ update time, whereas SCCs require $\Omega(n^{1.528})$ update time.

**Upper bounds.** Abboud and Vassilevska Williams [1] showed that in order to have $\widetilde{O}(n^{2-\epsilon})$ amortized update time for SC using fast matrix multiplication (FMM) is essential, thus ruling out non-trivial combinatorial approaches. And indeed, FMM-based algebraic dynamic reachability algorithms [20, 26] allow breaking the quadratic update bound. This is because

[1] Abboud and Vassilevska Williams[1] actually write that $\Omega(m^{1-o(1)})$ is the best bound one can hope for. However, since this bound is a function of $m$ exclusively, it is required to hold only for *some* density regime, e.g., sparse graphs.

[2] More specifically, based on their OMv conjecture variants, van den Brand, Nanongkai, and Saranurak [26] proved, for any $\epsilon > 0$, an $\Omega(\min_{a,b\in[0,1]}(n^{a+b} + n^{\omega(1,a,1)-a} + n^{\omega(1,a,b)-b}) \cdot n^{-\epsilon})$ lower bound for dynamic $s,t$-reachability, and an $\Omega(n^{1+\rho-\epsilon})$ lower bound for dynamic single-source reachability (SSR), where $\rho < 0.529$ satisfies $1 + 2\rho = \omega(1,\rho,1)$. Here $O(n^{\omega(a,b,c)})$ denotes the time needed to multiply an $n^a \times n^b$ boolean matrix by an $n^b \times n^c$ boolean matrix. These two conditional lower bounds become $\Omega(n^{1.406})$ and $\Omega(n^{1.528})$, respectively, assuming the current exponents of rectangular matrix multiplication. Furthermore, they become $\Omega(n^{1+1/4-\epsilon})$ and $\Omega(n^{1+1/2-\epsilon})$, respectively, assuming $\omega = 2$.

SC easily reduces to maintaining reachability to and from an arbitrary single vertex [26]. As a result, $O(n^{1.529})$ worst-case update time is possible in this case. This state-of-the-art bound does not match the $O(n^{1.406})$ lower bound [26], though. More generally, since the fully dynamic data structure of Sankowski [20] allows relatively cheap $O(n^{0.529})$-time arbitrary-pair reachability queries[3] at the cost of $O(n^{1.529})$ update time, this approach generalizes to maintaining some *at most* $k$ distinct SCCs of $G$ in $O(kn^{1.529})$ worst-case time per update. In particular, for $k = 2$ this trivially yields a solution to SC2 with $O(n^{1.529})$ worst-case update time. This matches the best known upper bound for SC. Consequently, the state-of-the-art bound for SC does not reflect the intuition of [1] that SC2 might be a harder problem.

The described approach, however, does not easily allow recomputing *all* SCCs of $G$, or even counting them, within subquadratic update time. In fact, the more general #SCC and SCCs problems do not seem to reduce to inspecting some $O(n^{2-\epsilon})$-sized subset of the reachability matrix of $G$. Nevertheless, Karczmarz, Mukherjee, and Sankowski [17] recently showed that SCCs (and thus #SCC) can be maintained in $O(n^{1.529})$ *amortized* time per update. Their approach is a mix of combinatorial and algebraic methods and critically depends on the efficiency of a *decremental SCCs* data structure. Bernstein, Probst Gutenberg, and Wulff-Nilsen [7] gave a near-optimal data structure maintaining SCCs *explicitly* under edge deletions with $\widetilde{O}(m)$ total update time. Although the $O(n^{1.529})$ amortized bound is near-optimal, the worst-case update time of the data structure of [7] might be $\Theta(n^2)$, and thus the worst-case update time of the fully dynamic SCCs data structure [17] may be as much as $\Theta(n^2)$ as well. To the best of our knowledge, no non-trivial worst-case bound for either #SCC or SCCs under fully dynamic edge updates has been described to date.

## 1.1 Our results

First of all, we close the gaps between the best known lower- and upper (worst-case) bounds for both fully dynamic strong connectivity (SC) and fully dynamic strongly connected components (SCC) in general directed graphs. See also Table 1. We achieve that by showing novel combinatorial reductions to the known fully dynamic arbitrary-pair reachability data structures [26, 20].

**Fully dynamic strongly connected components.** For fully dynamic SCCs we prove:

▶ **Theorem 1.** *Let $G$ be a digraph. Suppose there is a fully dynamic reachability data structure $\mathcal{D}$ processing single-edge updates and arbitrary-pair queries on $G$ in $U(n)$ and $Q(n)$ time respectively. Suppose the answers produced by $\mathcal{D}$ are correct with high probability*[4]*.*

*Then one can explicitly maintain the SCCs of $G$ subject to single-edge insertions and deletions in $\widetilde{O}(U(n) + n \cdot Q(n))$ time per update (worst-case if the bounds $U(n), Q(n)$ are worst-case). The obtained data structure is Monte Carlo randomized. The answers produced are correct with high probability.*

It is worth noting that the data structure trivially works against an adaptive adversary[5] since the revealed information, the SCCs, depend only on the current graph $G$. As far as the adversarial model of the assumed data structure $\mathcal{D}$ is concerned, no special requirements

[3] Throughout, we use the term *arbitrary-pair reachability* to refer to the situation when the endpoints $s, t$ of a reachability pair of interest are a part of a query and may vary for different queries. The output of a query is a single bit indicating whether $s$ can reach $t$ in the (current) graph.

[4] That is, with probability at least $1 - 1/n^c$, where the constant $c \geq 1$ can be set arbitrarily.

[5] That can only see the output of the data structure, and not the random bits used by the data structure internally.

beyond high-probability correctness are needed. Indeed, the query interface of $\mathcal{D}$ does not leak any information about the potential random choices made by $\mathcal{D}$ internally (unless $\mathcal{D}$ errs, which we treat as a low-probability failure anyway) since the correct answers are uniquely determined by the graph maintained in $\mathcal{D}$.

Sankowski [20] showed a Monte Carlo randomized data structure with $U(n) = O(n^{1+\rho})$ and $Q(n) = O(n^{\rho})$ (both worst-case), where $\rho < 0.529$ is such that $1 + 2\rho = \omega(1, \rho, 1)$. As a result, Theorem 1 implies a data structure with $O(n^{1.529})$ worst-case update bound for fully dynamic SCCs. To the best of our knowledge, we are the first to achieve a non-trivial $O(n^{2-\epsilon})$ *worst-case* update time for SCCs and even #SCC. Moreover, via a folklore[6] reduction from fully dynamic single-source reachability and the conditional lower bound of [26], the obtained $O(n^{1.529})$ bound is tight for SCCs. Hence, our data structure constitutes the final answer to the problem (up to polylog factors), unless the *Mv-hinted Mv* variant of the OMv conjecture of [26, Conjecture 5.7] fails. That being said, it is not clear whether our bound is tight for the counting variant #SCC whose output is a single number, as opposed to $n$ bits in SSR.

The worst-case update time guarantee allows applying Theorem 1 in the fault-tolerant model, where the goal is to recompute a graph property for a query set of at most $k$ failed edges or vertices. Georgiadis, Italiano, and Parotsidis [12] showed that after optimal linear preprocessing, one can compute the SCCs of $G$ after removing any single vertex/edge in optimal $O(n)$ time. Baswana, Choudhary, and Roditty [2] showed that after polynomial preprocessing, one can find the SCCs of $G$ after removing $k$ vertices/edges in $\widetilde{O}(2^k \cdot n)$ time. Thus, the known results could only handle at most $\log n$ failures faster than recompute-from-scratch, for any density of $G$. Our result implies that for $m = \Omega(n^{1.53})$ one can recompute SCCs faster than from scratch even under *polynomially* many (that is, $k = O(m/n^{1.529})$) vertex/edge failures[7]. That being said, the purely combinatorial data structures [2, 12] are significantly faster if, e.g., $k$ is constant.

It is worth noting that a reduction of computing SCCs to a reachability problem has been known in the parallel setting [21]. That static reduction does not seem to be applicable in the dynamic setting using algebraic techniques, though. More specifically, Schudy [21] reduces computing SCCs of $G$ to a number of adaptively generated instances of *multi-source reachability* (which in turn reduces to the single-source case by adding a super-source) on induced subgraphs of $G$ of total size $\widetilde{O}(m)$. In the dynamic setting, spending time linear in $m$ per update is prohibitive, and furthermore it is not clear how to efficiently simulate adaptively generated multi-source reachability queries for the following reasons. First, using the super-source trick would either require performing possibly $\Omega(n)$ edge updates (which are very costly, i.e., super-linear using algebraic methods) in the reachability data structure. If we wanted to avoid that, i.e., handle each source separately, we would possibly end up performing $\Omega(n^2)$ single-pair reachability queries. Our reduction for dynamic SCCs relies on being able to query $\widetilde{O}(n)$ arbitrary reachability pairs after each update. Crucially, these pairs cannot be grouped into, e.g., polylog$(n)$ multi-source queries. Such an $n$-arbitrary-pairs reachability problem looks much more difficult even in the static setting and does not seem to be solvable in linear time like multi-source reachability. In fact, we believe no static bound beyond $O(\min(nm, n^{\omega}))$ is known.

---

[6] It is enough to maintain a graph $G'$ obtained from $G$ by adding a super-sink $t$ and edges $vt$ for all $v \in V$. Then for a query source $s$, vertices reachable from $s$ in $G$ are those in the SCC of $s$ in $G' + ts$.

[7] Vertex failures can be easily simulated by splitting each $v \in V$ into $v_{\text{in}}, v_{\text{out}}$, that inherit the incoming and outgoing edges of $v$, resp, and introducing an edge $e_v = v_{\text{in}}v_{\text{out}}$. Then the failure of $v$ is equivalent to the failure of $e_v$.

**Strong connectivity.** For the less general dynamic strong connectivity (SC), we give an even more efficient and *deterministic* black-box reduction to dynamic arbitrary-pair reachability.

▶ **Theorem 2.** *Let $G$ be a digraph. Suppose there is a fully dynamic reachability data structure processing single-edge updates and arbitrary-pair reachability queries on $G$ in at most $T(n)$ time. Then one can maintain whether $G$ is strongly connected subject to single-edge insertions and deletions in $O(T(n)\log n)$ time per update (worst-case if the $T(n)$ bound is worst-case).*

van den Brand, Nanongkai, and Saranurak [26] showed a Monte Carlo randomized fully dynamic arbitrary-pair reachability data structure with $O(n^{1.407})$ worst-case update and query time. Therefore, Theorem 2 implies $O(n^{1.407})$ worst-case update bound for fully dynamic strong connectivity. This improves upon the known $O(n^{1.529})$ bound following from the trivial reduction to dynamic single-source reachability. By the conditional lower bound of [26], we essentially close the complexity of fully dynamic strong connectivity (up to polylogarithmic factors), unless the uMv-hinted uMv conjecture of [26, Conjecture 5.12] fails.

Whereas the obtained $O(n^{1.407})$ update bound is Monte Carlo randomized, this is only caused by the Schwartz-Zippel-lemma-style randomization inside the used reachability data structure. Obtaining a deterministic subquadratic fully dynamic reachability data structure would immediately imply a deterministic subquadaratic bound for SC.

Interestingly, our reduction *does not* seem to generalize to maintaining whether $G$ has more than two SCCs (SC2), which was the case for the trivial reduction. As a result, $O(n^{1.529})$ currently remains the sharpest update bound for SC2. At the same time the tightest conditional lower bound we have for SC2 is $\Omega(n^{1.407})$ because SC2 does not seem to be easily reducible from dynamic single-source reachability, which was the case for dynamic SCCs. This is a consequence of the fact that the output in the SCC problem is $n$ numbers, whereas in SC2 it is just a single bit. Obtaining a faster dynamic algorithm for SC2 or coming up with a plausible lower bound beyond $O(n^{1.407})$ is an interesting next step.

**Table 1** Comparison of state-of-the art conditional lower- and upper bounds for the problems SC, SC2, #SCC, and SCC, and new upper bounds obtained via our reductions. Each problem in the table generalizes the preceding one. For SC and SCC, our obtained upper bounds are tight.

<table>
<tr><th>problem</th><th>known conditional lower bound</th><th>prev. worst-case update bound</th><th>previous amortized update bound</th><th>our new worst-case update bound</th></tr>
<tr><td>SC</td><td rowspan="3">$\Omega(n^{1.406})$<br>[1]+[26]</td><td rowspan="2">$O(n^{1.529})$<br>[26, 20]</td><td rowspan="2">same as worst-case</td><td>$O(n^{1.407})$<br>Theorem 2+[26]</td></tr>
<tr><td>SC2</td><td>–</td></tr>
<tr><td>#SCC</td><td rowspan="2">$O(n^2)$<br>trivial</td><td rowspan="2">$O(n^{1.529})$<br>[17]</td><td rowspan="2">$O(n^{1.529})$<br>Theorem 1+[20]</td></tr>
<tr><td>SCC</td><td>$\Omega(n^{1.528})$<br>[26] via SSR</td></tr>
</table>

**Generalizations.** The reduction of Theorem 2 does generalize in two different ways. First, within the same update bound (see Theorem 11) we can actually maintain the exact size $\chi(G)$ of the *minimum strong connectivity augmentation* [10] of $G$. The minimum strong connectivity augmentation is a smallest possible set of edges that one needs to add to $G$ to make it strongly connected. $\chi(G)$ can be seen as an alternative and "orthogonal" (to the number of SCCs) measure of how much $G$ *is not* strongly connected. As shown by Eswaran and Tarjan [10], in the static setting, the value $\chi(G)$ and some minimum strong connectivity augmentation can be computed in linear time.

As is typical for algebraic data structures[8], maintaining an *object* – some minimum strong augmentation in our case – is trickier than maintaining only the optimal *value* $\chi(G)$. Our algorithm can be nevertheless extended to maintain some $\chi(G)$-sized augmentation $Y$ within the same $O(n^{1.407})$ worst-case bound and against an adaptive adversary, as long as $\chi(G) = O(n^{0.703})$. On the other hand, for larger $\chi(G)$, e.g., $\chi(G) = \Theta(n)$, some minimum augmentation $Y$ can be maintained in $O(n^{1.529})$ worst-case time per update.

Interestingly, the weighted version of the minimum strong connectivity augmentation problem, where the costs of different potential edges to be added vary, is NP-hard [10].

Moreover, the reduction behind Theorem 2 enables *universal* reachability queries, even if $G$ is not strongly connected. That is, in $O(n^{1.407})$ time, we can decide whether a query vertex $v$ can reach all vertices in $G$, or find some vertex that $v$ cannot reach. See Theorem 14. The $O(n^{1.529})$ query time follows trivially (simply issue all the $n$ possible reachability queries from $v$) from the fully dynamic single-source reachability data structure [20].

## 1.2 Further related work

Since the known lower bounds [1, 14] imply that fully dynamic (or partially dynamic with worst-case bounds) SCCs data structures are possible only for sufficiently dense graphs and using algebraic methods, most of the previous work regarding maintaining SCCs dealt with partially dynamic settings with the goal of optimizing total update time.

Maintaining SCCs has been studied in the incremental setting [5, 3, 13]. The current best known randomized solution for sparse graphs [5] has $\widetilde{O}(m^{4/3})$ total update time, whereas the best known deterministic data structure [3] has $O(m^{3/2})$ total update time. For denser graphs, Bender et al. [3] gave a near-optimal deterministic data structure with $\widetilde{O}(n^2)$ total update time. All these data structures maintain SCCs explicitly.

Decremental SCCs maintenance has been studied even more extensively [6, 7, 8, 18, 19]. Bernstein, Probst Gutenberg, and Wulff-Nilsen [7] gave a randomized data structure with near-optimal $\widetilde{O}(m)$ total update time. The current best known deterministic total update time bound of $\widetilde{O}(mn^{2/3})$ for general digraphs is due to Bernstein, Probst Gutenberg, and Saranurak [6]. For planar graphs, the near-optimal $\widetilde{O}(n)$ total update time can be achieved deterministically [16].

In the case of (global) strong connectivity, incremental graph search from/to an arbitrary single source yields optimal $O(m)$ total update time. It is not clear if decremental strong connectivity is easier, in any sense, than decremental SCCs, and no specialized data structures beyond those for SCCs maintenance [7, 6] are known for decremental strong connectivity.

# 2 Preliminaries

In this paper we deal with *directed* graphs. We write $V(G)$ and $E(G)$ to denote the sets of vertices and edges of $G$, respectively. We omit $G$ when the graph in consideration is clear from the context. A graph $H$ is a *subgraph* of $G$, which we denote by $H \subseteq G$, if and only if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. We write $e = uv \in E(G)$ when referring to edges of $G$. By $G^{\mathrm{R}}$ we denote $G$ with edges reversed.

A sequence of vertices $P = v_1 \dots v_k$, where $k \geq 1$ is called an $s \to t$ path in $G$ if $s = v_1$, $v_k = t$ and there is an edge $v_i v_{i+1}$ in $G$ for each $i = 1, \dots, k-1$. We sometimes view a path $P$ as a subgraph of $G$ with vertices $\{v_1, \dots, v_k\}$ and (possibly zero) edges $\{v_1 v_2, \dots, v_{k-1} v_k\}$.

[8] For example, for dynamic reachability and exact distances, the known algebraic data structures [25, 26, 20] are polynomially faster than the known solutions to their respective path-reporting variants [4, 17].

For convenience, we sometimes consider a single edge $uv$ a path. If $P_1$ is a $u \to v$ path and $P_2$ is a $v \to w$ path, we denote by $P_1 \cdot P_2$ (or simply $P_1P_2$) a path obtained by concatenating $P_1$ with $P_2$. A vertex $t \in V(G)$ is *reachable* from $s \in V(G)$ if there is an $s \to t$ path in $G$.

## 3 Fully dynamic strongly connected components

In this section we describe our fully dynamic strongly connected components data structure.

Suppose there exists a reachability data structure maintaining an $n$-vertex digraph subject to single-edge insertions and deletions and answering arbitrary-pair reachability queries. Denote by $U(n)$ and $Q(n)$ the update and query (resp.) time bounds of the assumed data structure.

We start by proving the following key observation, that tracking the vertices $v \in V$ strongly connected to *some* vertex $t$ out of a chosen subset $T \subseteq V$ can be reduced to maintaining some $n$ cells of the transitive closure matrix (reachability pairs) of a certain auxiliary graph $G_T$.

▶ **Lemma 3.** *Let $G = (V, E)$ be a digraph, and let $T \subseteq V$. Let $G'$ be a graph with vertices $V' = \{v' : v \in V\}$, and edges $E' = \{u'v' : uv \in E\}$, i.e., $G'$ is a copy of $G$ with vertices primed. Consider a graph $G_T$ obtained from $G \cup G'$ by adding edges $tt'$ for each $t \in T$. Then, for any $v \in V$, $v$ is strongly connected with* some *$t \in T$ in $G$ if and only if there is a path $v \to v'$ in $G_T$.*

**Proof.** Suppose that $v$ is strongly connected with some $t \in T$ in $G$. Then, there exists a path $P = v \to t$ in $G$, and a path $Q = t \to v$ in $G$. As a result, there is a corresponding path $Q' = t' \to v'$ in $G'$. Since $tt' \in E(G_T)$, we conclude there is a $v \to v'$ path $P \cdot tt' \cdot Q'$ in $G_T$.

Now suppose there is a $v \to v'$ path in $G_T$. Note that a path $R$ starting in the $G$-part of $G_T$ can only depart from $G$ for $G'$ through an edge $tt'$, $t \in T$, and it cannot go back to $G$. As a result, $R$ can be expressed as $R_1 \cdot (tt') \cdot R'_2$, where $R_1 = v \to t$ is a path in $G$, and $R'_2$ is a $t' \to v'$ path in $G'$. But $R'_2$ has a corresponding path $R_2 = t \to v$ in $G$. The paths $R_1, R_2$ certify that $v$ and $t$ are strongly connected. Consequently, $v$ is strongly connected to some vertex of $T$ in $G$. ◀

To make use of Lemma 3 we will employ the *unique witness trick* (usually attributed to [22]) that have been used for computing witnesses of various matrix products, also in the dynamic setting [4]. However, in our case, we will use this technique with a conceptually very different purpose: to *select* a certain *root* of every SCC that ever appears in $G$ throughout edge updates.

▶ **Lemma 4.** [22] *Let $X \subseteq V$ be a subset with $n/2^{k+1} \le |X| \le n/2^k$ for an integer $k \ge 0$. Let $S \subseteq V$ be a subset obtained by sampling $2^k$ elements of $V$ uniformly at random and independently (with replacement). Then, with probability at least $1/6$, $|X \cap S| = 1$.*

**Proof.** The probability that only the $i$-th sampled vertex is a unique element of $X \cap S$ is

$$\frac{|X|}{n} \cdot \left(1 - \frac{|X|}{n}\right)^{2^k - 1} \ge \frac{1}{2^{k+1}} \cdot \left(1 - \frac{1}{2^k}\right)^{2^k - 1} \ge \frac{1}{2^{k+1}e}.$$

The probability that $|X \cap S| = 1$ is at least the probability of one of the above disjoint events happening, i.e., at least $2^k \cdot \frac{1}{2^{k+1}e} = \frac{1}{2e} \ge 1/6$. ◀

Let $\ell = (\gamma + 2) \cdot \log_{6/5} n$ for a constant $\gamma > 0$ that can be adjusted to control the error probability. Let $B$ be the smallest integer such that $n < 2^B$, so that $B = O(\log n)$. For a non-negative integer $m$, put $[m] := \{0, \ldots, m\}$. Let us identify $V$ with the $B$-bit numbers $\{1, \ldots, n\}$. Moreover, let

$$\mathcal{T} = \{T_{k,l,b} : k \in [B-1], l \in [\ell - 1], b \in [B-1]\}$$

be a family of subsets of $V$ obtained as follows. For $k \in [B-1], l \in [\ell - 1]$ let $T_{k,l}$ be a subset of $V$ obtained by sampling $2^k$ elements of $V$ uniformly at random and independently, with replacement. Then, set $T_{k,l,b}$ to be a subset of numbers $v \in T_{k,l}$ such that the $b$-th (0-based) bit of $v$ in the binary equals 1. Clearly, the family $\mathcal{T}$ has $O(\log^3 n)$ subsets.

First of all, we maintain the graph $G$ itself in the assumed fully dynamic reachability data structure $\mathcal{D}$. For each $T \in \mathcal{T}$, we set up another assumed data structure $\mathcal{D}_T$ maintaining the graph $G_T$, as defined in Lemma 3. Note that each edge update to the graph $G$ is reflected using just two edge updates to each graph $G_T$ since $G_T$ consists of two copies of $G$. As a result, every edge update can be processed in $O(U(n) \cdot |\mathcal{T}|) = \widetilde{O}(U(n))$ time (worst-case if the bound $U(n)$ is worst-case).

We now describe how the SCCs of $G$ are recomputed after an update. We build an auxiliary graph $H$ whose construction follows. For each $v \in V$, $k \in [B-1]$, and $l \in [\ell - 1]$, we do the following. For every $b \in [B-1]$, we query the data structure $D_{T_{k,l,b}}$ whether there exists a path $v \to v'$ in $G_{T_{k,l,b}}$. Let $s_{v,k,l}$ be a number (vertex) such that the $b$-th bit is set to 1 if and only if the corresponding path in $G_{T_{k,l,b}}$ exists. If $s_{v,k,l} \in \{1, \ldots, n\}$ (so that $s_{v,k,l}$ is indeed a vertex of $G$) and $v$ *is strongly connected with* $s_{v,k,l}$ (which can be tested using two queries to $\mathcal{D}$), we add an undirected edge $\{v, s_{v,k,l}\}$ to $H$.

Observe that the graph $H$ is constructed by performing $\widetilde{O}(n)$ queries to the assumed data structures $\mathcal{D}$ and $\mathcal{D}_T$, $T \in \mathcal{T}$. The cost of these queries is $\widetilde{O}(n \cdot Q(n))$ (worst-case if $Q(n)$ is worst-case).

Having constructed the undirected graph $H$, we output its connected components as the SCCs of $G$. Since $H$ has $\widetilde{O}(n)$ edges, this can be done in $\widetilde{O}(n)$ additional worst-case time.

▶ **Lemma 5.** *With probability at least $1 - 1/n^\gamma$, the connected components of $H$ and the strongly connected components of $G$ are equal.*

**Proof.** By the construction, if two vertices $u, v \in V$ are connected by an edge (or more generally, a path) in $H$, then they are strongly connected in $G$. Hence, we only need to prove that if $u, v \in V$ are strongly connected in $G$, then they are connected in $H$ as well.

Let $C$ be the SCC of $G$ containing both $u$ and $v$. Let $z \in [B-1]$ be such an integer that we have $n/2^{z+1} \leq |C| \leq n/2^z$. By Lemma 4, for each $l \in [\ell - 1]$, $T_{z,l}$ satisfies $|T_{z,l} \cap C| = 1$ with probability at least $1/6$. As a result, with probability at least $1 - (5/6)^\ell \geq 1 - 1/n^{\gamma+2}$, at least one of the sets $T_{z,l}$, say $T_{z,y}$, satisfies $|T_{z,y} \cap C| = 1$. Let $s$ be the unique element of $T_{z,y} \cap C$. Observe that for $b \in [B-1]$, we have $T_{z,y,b} \cap C = \{s\}$ if the $b$-th bit of $s$ equals 1 and $T_{z,y,b} \cap C = \emptyset$ otherwise. As a result, by Lemma 3, for both $u$ and $v$ the query to the data structure $\mathcal{D}_{T_{z,y,b}}$ will return true if and only if the $b$-th bit of $s$ is 1. Equivalently, we will have $s = s_{v,k,l} = s_{u,k,l} \neq 0$. Since both edges $\{v, s\}$ and $\{u, s\}$ are added to $H$, $u$ and $v$ are indeed connected in $H$.

As there are at most $n^2$ pairs $(u, v)$, the implications hold with probability at least $1 - n^2 \cdot (1/n^{\gamma+2}) = 1 - 1/n^\gamma$. ◀

Note that the above lemma holds for any version of $G$ since, unless the algorithm makes a mistake (which happens with low probability), we do not reveal any random bits to the adversary as the output is always unique. To summarize, we obtain the following.

▶ **Theorem 1.** *Let $G$ be a digraph. Suppose there is a fully dynamic reachability data structure $\mathcal{D}$ processing single-edge updates and arbitrary-pair queries on $G$ in $U(n)$ and $Q(n)$ time respectively. Suppose the answers produced by $\mathcal{D}$ are correct with high probability.*

*Then one can explicitly maintain the SCCs of $G$ subject to single-edge insertions and deletions in $\widetilde{O}(U(n) + n \cdot Q(n))$ time per update (worst-case if the bounds $U(n), Q(n)$ are worst-case). The obtained data structure is Monte Carlo randomized. The answers produced are correct with high probability.*

Sankowski [20] showed that there exists a fully dynamic reachability data structure with $U(n) = \widetilde{O}(n^{1+\rho})$ worst-case update time and $Q(n) = \widetilde{O}(n^{\rho})$ query time, where $\rho < 0.529$ is such that $1 + 2\rho = \omega(1, \rho, 1)$. By Theorem 1 it follows that there exists a fully dynamic data structure maintaining SCCs explicitly with $\widetilde{O}(n^{1+\rho}) = O(n^{1.529})$ worst-case update time.

## 4 Reducing dynamic strong connectivity to dynamic reachability

In this section we show how to maintain whether the graph $G$ is strongly connected subject to edge insertions and deletions using a fully dynamic reachability data structure supporting single-edge updates and arbitrary-pair reachability queries. Our reduction is deterministic and incurs only polylogarithmic overhead. Again, assume that $n < 2^B$ for an integer $B = O(\log n)$. The vertices are numbered 1 through $n$ and thus can be seen as $B$-bit numbers.

Let us define a *source strongly connected component* of $G$ to be an SCC of $G$ with no incoming edges from other SCCs of $G$. Clearly, if $G$ is strongly connected, it has precisely one source SCC. The main idea is to maintain a *source set* $\mathcal{Z} \subseteq V$ such that $\mathcal{Z}$ contains *precisely one* vertex $z$ of every source SCC of $G$. For example, if $G$ is strongly connected, then $\mathcal{Z}$ can be any single-element subset of $V$. We also have the following simple property.

▶ **Observation 6.** *For every vertex $v \in V(G)$ there exists some source SCC $S$ of $G$ such that $v$ is reachable from every $s \in S$.*

**Proof.** Let $C$ be the SCC of $v$ in $G$. Let $G^*$ be the *condensation* of $G$ obtained from $G$ by contracting the SCCs. Then, $G^*$ is a DAG. In a DAG, every vertex is reachable from a source (that is, a vertex with no incoming edges). In particular, $C$ is reachable from some source $S$ of $G^*$. Consequently, any vertex $s \in S$ can reach $v$ in $G$ by construction of $G^*$. ◀

The following lemma shows how a data structure maintaining some source set of $G$ can be used to maintain strong connectivity of $G$.

▶ **Lemma 7.** *Let $G^R$ be the reverse of $G$. Let $\mathcal{Z}$ be some source set of $G$ and let $\mathcal{Z}^R$ be some source set of $G^R$. Then $G$ is strongly connected if and only if $|\mathcal{Z}| = |\mathcal{Z}^R| = 1$ and $t \in \mathcal{Z}^R$ can reach $s \in \mathcal{Z}$ in $G$.*

**Proof.** Clearly, if $G$ is strongly connected, then $\mathcal{Z} = \{s\}$, $\mathcal{Z}^{\mathrm{R}} = \{t\}$ for some $s, t \in V$. Moreover, $t$ can reach $s$ in $G$.

Now suppose $\mathcal{Z} = \{s\}$ and $\mathcal{Z}^{\mathrm{R}} = \{t\}$. Consider any $x, y \in V$. By Observation 6, $y$ is reachable from $s$. Similarly, by Observation 6 applied to $G^{\mathrm{R}}$, $x$ is reachable from $t$ in $G^{\mathrm{R}}$, i.e., $t$ is reachable from $x$ in $G$. By the assumption, $t$ can reach $s$ in $G$. Thus, there exist a path $x \to t \to s \to y$ in $G$. Since the pair $x, y$ was arbitrary, $G$ is indeed strongly connected. ◀

By Lemma 7, we can maintain strong connectivity using a fully dynamic source set data structure built on $G$, a fully dynamic source set data structure built on $G^{\mathrm{R}}$, and a fully dynamic reachability data structure built on $G$. Every edge update translates to a single edge update in the three data structures, and a single query to the reachability data structure.

Let us now focus on maintaining a source set $\mathcal{Z}$ of $G$ subject to edge updates. We have:

▶ **Lemma 8.** *For any $w \in V$, there exists at most one $z \in \mathcal{Z}$ such that $w$ can reach $z$ in $G$. Moreover, $w$ is in a source SCC (containing $z$) if and only if such a $z \in \mathcal{Z}$ exists.*

**Proof.** Let $z \in \mathcal{Z}$ and let $C$ be the source SCC of $G$ with $z \in C$. By the definition of a source SCC, the only vertices of $V$ that can reach $z$ are those in $C$. As a result, if $w$ can reach $z$, then $w \in C$. Therefore, $w \notin C'$ for other source SCCs $C' \neq C$, and thus $w$ cannot reach any other $z' \in \mathcal{Z}$, $z' \neq z$. On the other hand, if $w \in C$ for some source SCC $C$, then $w$ can clearly reach the unique element of $\mathcal{Z} \cap C$. ◀

**Maintained data structures.** First of all, let $\bar{G}$ be the graph $G$ extended with a super source $s \notin V(G)$ and edges $sz$ for all $z \in \mathcal{Z}$. We store $\bar{G}$ in a fully dynamic reachability data structure $\mathcal{D}$. Note that using a single reachability query to $\mathcal{D}$ we can check whether some query vertex $v \in V$ is reachable from $\mathcal{Z}$ in $\bar{G}$. Since $G \subseteq \bar{G}$ and $s$ has no incoming edges, the data structure $\mathcal{D}$ can also handle usual reachability queries in $G$.

For $b \in [B-1]$, let $G_b$ be the graph $G$ extended with a super sink $t \notin V(G)$, and edges $zt$ for all $z \in \mathcal{Z}$ such that the $b$-th bit of the number $z$ equals 1. We store the $O(\log n)$ graphs $G_b$ in fully dynamic reachability data structures $\mathcal{D}_0, \ldots, \mathcal{D}_{B-1}$. Note that since every $v \in V$ has at least one bit set, $\bigcup_{i=0}^{B-1} G_b$ contains edges $zt$ for all $z \in \mathcal{Z}$. As a result, a vertex $w \in V$ can reach some vertex of $\mathcal{Z}$ if and only if $w$ can reach $t$ in $\bigcup_{i=0}^{B-1} G_b$. Moreover, by Lemma 8, if $w$ can reach some vertex of $\mathcal{Z}$, then it can reach precisely one such $z \in \mathcal{Z}$. The individual data structures $G_b$ can be used to find that $z$: observe that $w$ can reach $t$ (equivalently, reach $z$) in $G_b$ precisely if and only if $z$ has the $b$-th bit set. As a result, we have the following.

▶ **Lemma 9.** *For any $w \in V$ one can find the unique $z \in \mathcal{Z}$ that $w$ can reach in $G$ (if one exists) using $O(\log n)$ queries to the fully dynamic reachability data structures $\mathcal{D}_0, \ldots, \mathcal{D}_{B-1}$.*

Whenever the graph $G$ is updated or the set $\mathcal{Z}$ undergoes an update, the update is passed to all the $O(\log n)$ relevant data structures $\mathcal{D}$ and $\mathcal{D}_0, \ldots, \mathcal{D}_{B-1}$ so that they reflect the current graphs $\bar{G}$ and $G_0, \ldots, G_{B-1}$ respectively.

We now proceed with describing how the updates are processed. In the following, denote by $G'$ the graph $G$ after the edge update, whereas $G$ denotes the graph before the edge update considered.

**Edge insertions.** First consider an insertion of edge $uv$. If there exists a path $u \to v$ in $G$ (a single query to $\mathcal{D}$), the SCCs of $G$ do not change due to the insertion. No source SCC gets an incoming edge from another SCC. Therefore, $\mathcal{Z}$ remains a valid source set.

Let us thus assume that there is no $u \to v$ path in $G$. In particular, $u$ and $v$ are not strongly connected in $G$. Let us denote by $S_v$ the SCC containing $v$ in $G$. Inserting $uv$ cannot break any of the existing SCCs of $G$, but might cause some SCCs of $G$ (in particular $S_v$) merge in $G'$. Let $S^*$ denote the SCC of $v$ in $G'$. We have $S_v \subseteq S^*$, but potentially $S^* = S_v$.

Let us now argue that $S_v$ is the only SCC of $G$ that might lose the status of a source SCC due to the insertion. Indeed, if $S_v \neq S^*$, then every other SCC $S' \neq S_v$ of $G$, such that $S' \subset S^*$, is reachable from $S_v$ since the insertion of $uv$ makes $S_v$ and $S'$ strongly connected. As a result, $S'$ is not a source SCC of $G$. Moreover, the respective sets of incoming inter-SCC edges of other SCCs $S''$ of $G$ such that $S'' \cap S^* = \emptyset$ do not change, so such $S''$ is a source SCC of $G'$ if and only if $S''$ is a source SCC of $G$.

By Lemmas 8 and 9 we can test whether $S_v$ is a source SCC (and possibly find the unique $z \in S_v \cap \mathcal{Z}$) by performing $O(\log n)$ queries to the data structures $\mathcal{D}_0, \ldots, \mathcal{D}_{B-1}$. The next step is to remove $z$ (if it exists) from $\mathcal{Z}$, and update all the data structures $\mathcal{D}, \mathcal{D}_0, \ldots, \mathcal{D}_b$ accordingly. Next, we apply the insertion of $uv$ to all these data structures so that they store

(supergraphs) of the updated graph $G'$. Observe that at this point the set $\mathcal{Z}$ might miss at most one element, if $S^*$ turns out to be a source SCCs of $G'$. In order to detect whether this is the case we check whether $v \in S^*$ can be reached from $\mathcal{Z}$ in $G'$. Recall that this amounts to a single query to $\mathcal{D}$ about the existence of a path from $s$ to $u$ in $\bar{G}$. If so, $S^*$ is reachable from some other SCC, and thus is not a source SCC. Otherwise, $S^*$ is indeed a source SCC and thus we add to $\mathcal{Z}$ an arbitrary element of $S^*$, e.g., the vertex $v$, so that $\mathcal{Z}$ again becomes a valid source set of $G'$.

**Edge deletions.** Edge deletions can be handled essentially symmetrically to edge insertions. Suppose an edge $uv \in E(G)$ is deleted. Again, if the deletion does not make $v$ unreachable from $u$ (which can be tested by performing a single update and query on $\mathcal{D}$), neither the SCCs of $G$ nor the source SCCs of $G$ change. So suppose $u$ cannot reach $v$ after the deletion.

Let $S'_v$ be the SCC containing $v$ in $G'$ and let $S^*$ be the SCC containing $v$ in $G$. We have $S'_v \subseteq S^*$ and potentially $S'_v = S^*$. If $S^*$ is a source SCC in $G$, we can find $z \in S^* \cap \mathcal{Z}$ using $O(\log n)$ reachability queries from $v$ to $t$ in $\mathcal{D}_0, \ldots, \mathcal{D}_{B-1}$ by Lemmas 8 and 9.

Observe that $S'_v$ is the only SCC contained in $S^*$ that might potentially be a source SCC of $G'$: if $S' \neq S'_v$ is an SCC of $G'$, and $S' \subset S^*$, then $S'$ is surely reachable from $S'_v$ in $G'$ and thus $S'$ is not a source SCC of $G'$. Similarly as we did when processing an insertion, we remove $z$ (if exists) from $\mathcal{Z}$. At this point, $\mathcal{Z}$ might need inserting at most one element in order to become a valid source set of $G'$. This is the case precisely when $S'_v$ is a source SCC of $G'$. We can check that, again, by issuing a single $s \to v$ reachability query to $\mathcal{D}$ after the deletion is reflected in $\mathcal{D}$. If $S'_v$ happens to be a source SCC of $G'$, we add $v \in S'_v$ to $\mathcal{Z}$.

Note that processing an edge update causes at most two element updates to the set $\mathcal{Z}$. Each of these updates, and also the edge update itself, is reflected in $O(\log n)$ fully dynamic reachability data structures $\mathcal{D}, \mathcal{D}_0, \ldots, \mathcal{D}_{B-1}$. Moreover, only $O(\log n)$ reachability queries are performed on these data structures. Consequently, we obtain the following.

▶ **Lemma 10.** *Suppose there is a fully dynamic reachability data structure processing single-edge updates and arbitrary-pair reachability queries on $G$ in at most $T(n)$ time. Then one can maintain a source set $\mathcal{Z}$ of $G$ explicitly subject to single-edge insertions and deletions in $O(T(n) \log n)$ time per update (worst-case if the $T(n)$ bound is worst-case).*

By Lemma 10 and the discussion after Lemma 7, we obtain:

▶ **Theorem 2.** *Let $G$ be a digraph. Suppose there is a fully dynamic reachability data structure processing single-edge updates and arbitrary-pair reachability queries on $G$ in at most $T(n)$ time. Then one can maintain whether $G$ is strongly connected subject to single-edge insertions and deletions in $O(T(n) \log n)$ time per update (worst-case if the $T(n)$ bound is worst-case).*

The currently best known bound on the maximum of query time and update time of a fully dynamic arbitrary-pair reachability data structure is $T(n) = O(n^{1.407})$ worst-case (Monte Carlo randomized) due to [26]. Consequently, Theorem 2 combined with [26] yields $O(n^{1.407})$ worst-case update time for fully dynamic strong connectivity.

**Strong connectivity augmentation.** Due to a result of Eswaran and Tarjan [10], using Lemma 10 we can actually prove a more general result.

▶ **Theorem 11.** *Let $G$ be a digraph. Let $T(n)$ be defined as in Theorem 2. Then one can maintain the size $\chi(G)$ of a minimum strong connectivity augmentation of $G$ subject to single-edge insertions and deletions in $O(T(n) \log n)$ time per update. (worst-case if the $T(n)$ bound is worst-case).*

**Proof.** Let $\mathcal{Z}$ and $\mathcal{Z}^{\mathrm{R}}$ be source sets of $G$ and $G^{\mathrm{R}}$ respectively. Define a *sink SCC* to be an SCC of $G$ that is a source SCC in $G^{\mathrm{R}}$. Eswaran and Tarjan [10] prove that if $G$ is not strongly connected, then $\chi(G)$ equals $\max(s,t)+q$, where $s$ denotes the number of source SCCs that are not sink SCCs, $t$ denotes the number of sink SCCs that are not source SCCs, and $q$ denotes the number of "isolated SCCs" that are both source and sink SCCs. If $G$ is strongly connected then $\chi(G)=0$ holds trivially.

Note that with $s,t,q$ defined like this, we have $|\mathcal{Z}|=s+q$ and $|\mathcal{Z}^{\mathrm{R}}|=t+q$. Hence, if $G$ is not strongly connected, then $\max(|\mathcal{Z}|,|\mathcal{Z}^{\mathrm{R}}|)=\max(s,t)+q=\chi(G)$. As a result, it is enough to maintain some sets $\mathcal{Z}$ and $\mathcal{Z}^{\mathrm{R}}$, and test whether $G$ is strongly connected using Lemmas 10 and 7. Clearly, we need only $O(T(n)\log n)$ time for this. ◀

Again, by combining Theorem 11 with [26], we obtain that the size $\chi(G)$ of a minimum strong connectivity augmentation can be maintained in $O(n^{1.407})$ worst-case time per update.

Theorem 11 does not immediately yield any actual set $Y$ of edges such that $|Y|=\chi(G)$ and $G+Y$ is strongly connected. We now discuss how such an augmentation $Y$ can be maintained. To this end we now sketch the method [10] of obtaining an augmentation with $\max(|\mathcal{Z}|,|\mathcal{Z}^R|)$ edges in case $G$ is not strongly connected. Observe that in such a case, the quantity $\max(|\mathcal{Z}|,|\mathcal{Z}^R|)$ is clearly a lower bound on the size of $Y$.

Suppose wlog. that $|\mathcal{Z}|\geq|\mathcal{Z}^R|$. For each $z\in\mathcal{Z}$, let $t_z\in\mathcal{Z}^R$ be arbitrary such that $z$ can reach $t_z$ in $G$; note that at least one such $t_z$ always exists. Similarly, for each $z'\in\mathcal{Z}^R$, let $s_{z'}\in\mathcal{Z}$ be arbitrary such that $z'$ is reachable from $s_{z'}$ in $G$.

Consider a set of edges $F=\{zt_z : z\in\mathcal{Z}\}\cup\{s_{z'}z' : z'\in\mathcal{Z}^R\}$. Let $M$ be a maximal matching in $F$ (possibly allowing self-loops), that is, a maximal subset $\{y_1y'_1,\ldots,y_ky'_k\}\subseteq F$ such that all $y_1,\ldots,y_k$ are distinct and all $y'_1,\ldots,y'_k$ are distinct.

Put $A:=\{y_1,\ldots,y_k\}$, and $B=\{y'_1,\ldots,y'_k\}$. Moreover, put $\mathcal{Z}\setminus A=\{z_1,\ldots,z_p\}$ and $\mathcal{Z}'\setminus B=\{z'_1,\ldots,z'_q\}$, where $p\geq q$. By the maximality of $M$, we have that for all $z\in\mathcal{Z}\setminus A$, $t_z\in B$, that is, $z$ can reach some vertex from $B$ in $G$. Similarly, for all $z'\in\mathcal{Z}^R\setminus B$, $z'$ is reachable from a vertex of $A$.

Put $y_{k+1}:=y_1$. Consider an augmentation:

$$Y=\{y'_iy_{i+1} : i=1,\ldots,k\}\cup\{z'_{\max(i,q)}z_i : i=1,\ldots,p\}.$$

We have $|Y|=k+p=|\mathcal{Z}|=\max(|\mathcal{Z}|,|\mathcal{Z}^R|)$. Eswaran and Tarjan [10] argue rather easily that $G+Y$ is strongly connected: (1) the source and sink vertices in $A\cup B$ are pairwise strongly connected since they are arranged in a cycle, (2) every vertex in $\mathcal{Z}^R\setminus B$ can reach $A\cup B$ via $\mathcal{Z}\setminus A$ and the edges in $Y$, and (3) every vertex in $\mathcal{Z}\setminus A$ can be reached from $A\cup B$ via $\mathcal{Z}^R\setminus B$ and the edges in $Y$.

The above construction reduces, in $O(n)$ time, the problem of maintaining an optimal $Y$ to maintaining:

**(1)** for each vertex $z\in\mathcal{Z}$, an arbitrary vertex $t_z\in\mathcal{Z}^R$ reachable from $z$ in $G$,
**(2)** for each vertex $z'\in\mathcal{Z}^R$, an arbitrary vertex $s_{z'}\in\mathcal{Z}$ that can reach $z'$ in $G$.

If $\chi(G)=\max(|\mathcal{Z}|,|\mathcal{Z}^R|)$ is small enough, we can achieve the above by simply maintaining the $(\mathcal{Z}\cup\mathcal{Z}^R)\times(\mathcal{Z}\cup\mathcal{Z}^R)$ submatrix of the transitive closure of $G$. van den Brand, Forster, and Nazari [25] showed the following.

▶ **Theorem 12.** [25] *Let $G$ be a directed graph. Let $S$ be a dynamic subset of $V$ be such that $|S|=O(n^{0.85})$ at all times. There exists a data structure explicitly maintaining reachability between all-pairs of vertices in $S$ subject to fully dynamic single-edge updates to $G$ and single-element additions/deletions to $S$ in $O(n^{1.407}+|S|^2)$ worst-case time per update.*

Recall that if $G$ is subject to fully dynamic single-edge updates, then $\mathcal{Z}$ and $\mathcal{Z}^R$ undergo at most two element updates per update to $G$. As a result, by putting $S = \mathcal{Z} \cup \mathcal{Z}^R$ in the above theorem and combining with Lemma 10, we can maintain a minimum strong connectivity augmentation $Y$ in $O(n^{1.407})$ time as long as $\chi(G)^2 = O(n^{1.407})$.

If $\chi(G) = \Theta(n)$, however, the above method would have quadratic update time. We now argue that if we expect $\chi(G)$ to be large, then some desired sets $\{t_z \in \mathcal{Z}^R : z \in \mathcal{Z}\}$, and $\{s_{z'} \in \mathcal{Z} : z' \in \mathcal{Z}^R\}$ can be maintained in $O(n^{1.529})$ worst-case time per update anyway. This is possible using the below simple *existential reachability* data structure.

▶ **Lemma 13.** *Let $G$ be a directed graph. Suppose there is a fully dynamic reachability data structure processing single-edge updates and arbitrary-pair reachability queries on $G$ in $U(n) = \mathrm{poly}(n)$ and $Q(n) = \mathrm{poly}(n)$ time respectively.*

*Let $X \subseteq V$ be a dynamic subset. There is a data structure maintaining $G$ subject to fully dynamic edge updates and $X$ subject to single-element insertions and deletions with $\widetilde{O}(U(n))$ update time supporting the following queries in $\widetilde{O}(Q(n))$-time. Given a query vertex $s \in V$, find some vertex $x \in X$ that $s$ can reach in $G$ (if such an $x$ exists).*

**Proof sketch.** Suppose wlog. that the vertices of $G$ are identified with $\{1, \ldots, n\}$ and $n$ is a power of two. Let $G'$ be obtained from $G$ as follows. First, we augment $G$ with an auxiliary full binary tree $\mathcal{T}$ with $n$ leaves labeled $l_1, \ldots, l_n$ from left to right, so that $V(\mathcal{T}) \cap V(G) = \emptyset$. The edges in $\mathcal{T}$ are directed from children to their parents. Second, for every $x \in X$, we add an edge $xl_x$ to $G'$. Note that the graph $G'$ has at most $3n$ vertices.

We maintain the graph $G'$ using the assumed reachability data structure $\mathcal{D}$. Whenever $G$ is subject to an edge insertion or deletion, that update is also applied to $G'$. Similarly, if an element $x$ is inserted/deleted from $X$, this is reflected in $G'$ by adding/deleting the edge $xl_x$.

The binary tree $\mathcal{T}$ allows locating some (or even the minimum-label) vertex of $X$ that a query vertex $s$ can reach within $O(\log n)$ queries to $\mathcal{D}$. Indeed, note that $s$ can reach a vertex $w \in V(\mathcal{T})$ in $G'$ if and only if $s$ can reach in $G$ some vertex $x \in X$ such that $w$ is an ancestor of the leaf $l_x \in V(\mathcal{T})$. In particular, $s$ can reach $X$ in $G$ if and only if $s$ can reach the root of $\mathcal{T}$ in $G'$. If this is the case, the search starts in the root of $\mathcal{T}$ and descends down the tree maintaining the invariant that one of the leaves in the current subtree is reachable from $s$ in $G'$. Once we reach a leaf $l_x$, we report $x \in X$ as reachable from $s$. ◀

With Lemma 13 in hand (applied twice, to the graph $G$ with $X := \mathcal{Z}^R$ and to the graph $G^R$ with set $X := \mathcal{Z}$), using fully dynamic reachability data structure of [20] with $O(n^{1.529})$ worst-case update time and $O(n^{0.529})$ query time, we can find the desired sets by issuing $O(n)$ existential reachability queries. Recall that the sets $\mathcal{Z}$ and $\mathcal{Z}^R$ undergo $O(1)$ updates per edge update to $G$, so the worst-case update time is $O(n^{1.529})$.

Finally, we note that the described algorithms for maintaining the minimum strong connectivity augmentation $Y$ work against an adaptive adversary. Indeed, they are deterministic if provided with the required reachability information about $G$. That in turn is uniquely defined and maintained correctly with high probability by the data structures of [25, 20].

**Universal reachability.** By maintaining the source set $\mathcal{Z}$, we can also achieve the following.

▶ **Theorem 14.** *Let $G$ be a digraph. Let $T(n)$ be defined as in Theorem 2. There exists a data structure maintaining $G$ under single-edge insertions and deletions and answering queries whether a given vertex $v$ can reach all vertices in $G$ (and if not, providing an example of an unreachable vertex) with $\widetilde{O}(T(n))$ update and query time.*

**Proof.** Let $\mathcal{Z}$ be again a source set of $G$. Clearly, if $|\mathcal{Z}| > 1$, then no vertex of $G$ can reach all other vertices. Otherwise, if $|\mathcal{Z}| = 1$, then some $v$ can reach all of $V$ if and only if $v$ can reach (or, equivalently, is strongly connected to) the only element $z \in \mathcal{Z}$. As a result, given $\mathcal{Z}$ and the data structure $\mathcal{D}$, we can check whether $v$ can reach all vertices in $G$ by issuing a $v \to z$ query to $\mathcal{D}$ in $\widetilde{O}(T(n))$ time. If $v$ cannot reach some vertex of $G$, then it cannot reach some $z' \in \mathcal{Z}$. Since there is at most one $z \in \mathcal{Z}$ that $v$ can reach, to find some unreachable $z' \in \mathcal{Z}$, it is enough to issue two queries to $\mathcal{D}$. ◀

## References


1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

2 Surender Baswana, Keerti Choudhary, and Liam Roditty. An efficient strongly connected components algorithm in the fault tolerant model. *Algorithmica*, 81(3):967–985, 2019. doi:10.1007/s00453-018-0452-3.

3 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, and Robert E. Tarjan. A new approach to incremental cycle detection and related problems. *ACM Trans. Algorithms*, 12(2):14:1–14:22, 2016. doi:10.1145/2756553.

4 Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1836–1855. SIAM, 2021. doi:10.1137/1.9781611976465.110.

5 Aaron Bernstein, Aditi Dudeja, and Seth Pettie. Incremental SCC maintenance in sparse graphs. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 14:1–14:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.14.

6 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1123–1134. IEEE, 2020. doi:10.1109/FOCS46700.2020.00108.

7 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 365–376. ACM, 2019. doi:10.1145/3313276.3316335.

8 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Łącki, and Nikos Parotsidis. Decremental single-source reachability and strongly connected components in $\tilde{o}(m\sqrt{n})$ total update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016*, pages 315–324. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.42.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

10 Kapali P. Eswaran and Robert Endre Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976. doi:10.1137/0205044.

11 Harold N. Gabow. Path-based depth-first search for strong and biconnected components. *Inf. Process. Lett.*, 74(3-4):107–114, 2000. doi:10.1016/S0020-0190(00)00051-X.

12 Loukas Georgiadis, Giuseppe F. Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. *SIAM J. Comput.*, 49(5):865–926, 2020. doi:10.1137/19M1258530.

**13** Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert Endre Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Trans. Algorithms*, 8(1):3:1–3:33, 2012. doi:10.1145/2071379.2071382.

**14** Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

**15** Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

**16** Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017*, pages 1108–1121. ACM, 2017. doi:10.1145/3055399.3055480.

**17** Adam Karczmarz, Anish Mukherjee, and Piotr Sankowski. Subquadratic dynamic path reporting in directed graphs against an adaptive adversary. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1643–1656. ACM, 2022. doi:10.1145/3519935.3520058.

**18** Jakub Łącki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algorithms*, 9(3):27:1–27:15, 2013. doi:10.1145/2483699.2483707.

**19** Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.

**20** Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science FOCS 2004*, pages 509–517. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.25.

**21** Warren Schudy. Finding strongly connected components in parallel using $o(\log^2 n)$ reachability queries. In *SPAA 2008: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 146–151. ACM, 2008. doi:10.1145/1378533.1378560.

**22** Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995. doi:10.1006/jcss.1995.1078.

**23** M. Sharir. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications*, 7(1):67–72, 1981. doi:10.1016/0898-1221(81)90008-0.

**24** Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

**25** Jan van den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 1011–1022. IEEE, 2022. doi:10.1109/FOCS54457.2022.00099.

**26** Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.

**27** Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

# An Improved Approximation Algorithm for the Max-3-Section Problem

**Dor Katzelnick**
The Henry and Marilyn Taub Faculty of Computer Science, Technion, Haifa, Israel

**Aditya Pillai**
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

**Roy Schwartz**
The Henry and Marilyn Taub Faculty of Computer Science, Technion, Haifa, Israel

**Mohit Singh**
H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

## Abstract

We consider the Max-3-Section problem, where we are given an undirected graph $G = (V, E)$ equipped with non-negative edge weights $w : E \to \mathbb{R}_+$ and the goal is to find a partition of $V$ into three equisized parts while maximizing the total weight of edges crossing between different parts. Max-3-Section is closely related to other well-studied graph partitioning problems, e.g., Max-Cut, Max-3-Cut, and Max-Bisection. We present a polynomial time algorithm achieving an approximation of 0.795, that improves upon the previous best known approximation of 0.673. The requirement of multiple parts that have equal sizes renders Max-3-Section much harder to cope with compared to, e.g., Max-Bisection. We show a new algorithm that combines the existing approach of Lassere hierarchy along with a random cut strategy that suffices to give our result.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Theory of computation → Semidefinite programming

**Keywords and phrases** Approximation Algorithms, Semidefinite Programming, Max-Cut, Max-Bisection

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.69

**Related Version** *Full Version*: https://arxiv.org/abs/2308.03516

**Funding** Mohit Singh and Aditya Pillai were supported in part by NSF CCF-2106444 and NSF CCF-1910423. Dor Katzelnick and Roy Schwartz receive funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 852870-ERC-SUBMODULAR.

**Acknowledgements** The authors would like to thank Uri Zwick for insightful discussions.

## 1 Introduction

In this paper we study the Max-3-Section problem: given an undirected graph $G = (V, E)$ equipped with non-negative edge weights $w : E \to \mathbb{R}_+$ the goal is to partition the vertex set $V$ into three equisized parts while maximizing the total weight of edges that cross between different parts. Max-3-Section is closely related to other classic graph partitioning problems, where given the same input as in Max-3-Section the goal is to output a partition of the vertex set $V$ (possibly given some problem specific constraint) while maximizing the total weight of edges that cross between different parts. Perhaps the most famous of these problems is Max-Cut, whose constraint is that the partition contains only two parts with no restriction on the size of these parts. Max-Cut is one of Karp's 21 NP-hard problems [11]


and in their seminal work Goemans and Williamson [9] presented an approximation of 0.8786 using semi-definite programming and random hyperplane rounding. It is known that the latter result is tight, assuming the unique games conjecture, as proved by Khot, Kindler, Mossel, and O'Donnell [12].

A natural problem that generalizes Max-Cut is Max-$k$-Cut, whose constraint is that the partition contains $k$ parts with no restriction on the size of these parts. A simple algorithm that returns a uniform random solution, i.e., every vertex is assigned independently and uniformly to one of the $k$ parts, achieves an approximation of $1 - 1/k$. Several works aimed at improving the guarantee of this simple algorithm, e.g., [6, 8, 4, 14]. For example, a notable case that attracted much attention is the Max-3-Cut problem [6, 8, 4, 14], whose best approximation is 0.836 and was given by Goemans and Williamson [8]. Interestingly, the latter guarantee is worse than the approximation known for Max-Cut. For general values of $k$, it was shown by Frieze and Jerrum [6] that an approximation of $1 - 1/k + \Theta(\ln k/k^2)$ can be achieved. Further improvements for the approximation guarantee were presented by de Klerk, Pasechnik and Warners [4].

Adding a single global constraint to Max-Cut that requires both parts to be of equal size leads to the classic problem of Max-Bisection. Max-Bisection was extensively studied throughout the years, e.g., [6, 16, 10, 5, 15, 2]. This sequence of works currently culminates with the works of Raghavendra and Tan [15], who present an approximation of 0.85 that is based on rounding a Lasserre hierarchy semi-definite program, which was subsequently improved to 0.877 by Austrin, Benabbas, and Georgiou [2] who improved the former rounding. The question whether one can obtain for Max-Bisection the same approximation guarantee that is known for Max-Cut remains a tantalizing open problem.

Both Max-3-Section and Max-Bisection are captured by the Max-$k$-Section problem [1, 7, 13, 3], which falls in the above broad family of graph partitioning problems and whose constraint is that the partition contains $k$ equisized parts. Similarly to Max-$k$-Cut, it is known that the simple algorithm that returns a uniform random solution achieves an approximation of $1 - 1/k$. Thus, it is no surprise that the goal of past research, e.g., [1, 7, 13], was to improve upon this guarantee. For the special case of Max-3-Section, Ling [13] presented an approximation of 0.6733 that is based on rounding a semi-definite programming relaxation similarly to Max-3-Cut [8]. For general values of $k$, Andersson [1] presented an approximation of $1 - 1/k + \Theta(1/k^3)$, again by rounding a suitable semi-definite program relaxation. Additionally, Gaur, Krishnamurti, and Kohli [7] presented a local search algorithm for a more general problem in which each part has a (possibly different) limit on its size.

The main focus of this work is the Max-3-Section problem. For $k = 2$, the best known approximation for Max-Bisection (which is essentially Max-Cut with a single global constraint on the size of the first part in the partition) equals 0.877 and is (almost) identical to the best possible approximation of 0.878 for Max-Cut. However, when $k = 3$, the best known approximation for Max-3-Section (which is essentially Max-3-Cut with two global constraints on the size of the first two pieces in the partition) equals 0.6733 and is far from the best known approximation of 0.836 for Max-3-Cut. Moreover, the former approximation guarantee of 0.6733 for Max-3-Section only slightly improves upon the $2/3$ approximation guarantee of the trivial algorithm that simply returns a uniform random solution. Thus, we aim to understand and minimize the gap that exists for $k = 3$.

## 1.1 Our Results and Techniques

We present the following main algorithmic result for the Max-3-Section problem:

▶ **Theorem 1.** *There is a polynomial time algorithm for Max-3-Section that achieves an approximation guarantee of at least* 0.795.

It is important to note that the approximation guarantee of the above theorem improves upon the previous best known algorithm for Max-3-Section [13] that achieves an approximation of 0.6773. As proving the exact approximation guarantee of our algorithm is an involved task (for reasons that will be clear soon), we also present the following conjecture that is based on numerical evidence:

▶ **Conjecture 2.** *The algorithm presented to prove Theorem 1 achieves an approximation of* 0.8192 *for Max-3-Section.*

We further show how the algorithm we present for proving Theorem 1 can be extended to general values of $k$. First, we prove that the algorithm, alongside its analysis, cannot provide an improved approximation as $k$ increases (when compared to its approximation for the case of $k = 3$). Second, using numeric evidence we conjecture that the approximation of the algorithm remains 0.8192 for $k$ ranging from 3 up to 6. Hence, the above gives rise to the following extended conjecture:

▶ **Conjecture 3.** *There is a polynomial time algorithm achieving an approximation of* 0.8192 *for Max-k-Section for* $k = 3, 4, 5$.

The above conjecture provides, for $k = 4, 5$, an improved approximation when compared to the previous best known result. The current best is a small improvement over the trivial randomized approximation algorithm by Andersson [1] which achieves an approximation of $1 - 1/k + \Theta(k^{-3})$.

When considering our approach to Max-3-Section we focus on two of its closely related problems: Max-3-Cut and Max-Bisection and examine the approaches used to design and analyze algorithms for both. Let us start with Max-3-Cut. The approach used to obtain the current best known algorithms for Max-3-Cut, e.g., [8, 4, 14], is based on the random hyperplane rounding method due to Goemans and Williamson [9] (as well as extensions of this method) of a semi-definite programming relaxation. This approach, when applied to Max-3-Section, suffers a significant drawback since it does not preserve marginal values (a marginal value is the likelihood the relaxation assigns to the event that vertex $u$ belongs to part $i$). Specifically, the expected number of vertices assigned to every part by the rounding algorithm might be incomparable to $n/3$ and therefore applying these ideas as was done by Ling [13] leads only to a minor improvement over the random solution algorithm. It is worth noting that this drawback is already present when considering Max-Bisection.

Let us now consider Max-Bisection, and specifically we focus on the approach of Raghavendra and Tan [15] (as well as Austrin, Benabbas, and Georgiou [2] who build upon [15]). First, a Lasserre hierarchy of a natural semi-definite programming relaxation for Max-Bisection is solved. Second, a rounding procedure that preserves the marginal values is applied to obtain a subset $C_1 \subseteq V$ of vertices such that: (1) there are sufficiently many edges crossing the cut $C_1$ defines; and (2) $C_1$ contains (roughly) $n/2$ vertices. Third, the solution is re-balanced to obtain a perfect bisection, i.e., ensuring that $|C_1| = n/2$ without a significant loss in the number of edges crossing between $C_1$ and $C_2 = V \setminus C_1$.

There are two main difficulties when considering this approach in the context of Max-3-Section. The first difficulty stems from the fact that we have three parts in Max-3-Section, whereas in Max-Bisection there are only two parts. Therefore, if we use the above rounding procedure to find $C_1 \subseteq V$ of size (roughly) $n/3$ with sufficiently many edges crossing the cut $C_1$ defines, it is not clear how to recurse and further partition $V \setminus C_1$. We note that in Max-Bisection no recursion is needed since $C_2$ is chosen to be $V \setminus C_1$. However, in Max-3-Section $V \setminus C_1$ still needs to be partitioned into $C_2$ and $C_3$. Our solution to this

difficulty is to *condition* the marginal values of vertices remaining in $V \setminus C_1$ on the fact that each remaining vertex was not chosen to $C_1$. Such a conditioning, intuitively, ensures that we preserve marginal values overall while recursing on $V \setminus C_1$.

The second difficulty in applying this apporach arises from the following observation: we can show that if one first creates part $C_1$, and then creates part $C_2$ (and part $C_3$ is all remaining vertices), then if the analysis is performed edge-by-edge (as is the case in both [15, 2]) no approximation better than 0.7192 can be achieved. Specifically, we present a configuration of the vectors that correspond to the endpoints of an edge that satisfy: (1) the vectors are feasible for the semi-definite programming relaxation; and (2) the ratio between the probability of this edge being separated by the rounding algorithm and the contribution of its vectors to the objective function of the relaxation is at most 0.7192 (refer to Section 4 for a formal definition of a configuration and to Observation 35 in the full version of the paper). An approximation of 0.7192, if possible given the above approach, improves the current best known approximation of 0.6733 [13]. However, we aim for a much larger improvement. Our solution to this difficulty is to *uniformly permute* the order in which the parts are generated. Since the approach based on [15, 2] preserves marginals, this permutation allows us to better cope with problematic configurations. It is important to note that a permutation is meaningless for Max-Bisection, since whether a vertex belongs to $C_1$ immediately implies whether it belongs to $C_2$ and vice versa. In Max-3-Section this is obviously not the case.

Thus, following the above discussion, our approach builds upon the approach for Max-Bisection with two added ingredients. The first is altering the marginal values the semi-definite programming relaxation provides via appropriate conditioning when recursing. The second is uniformly permuting the order in which the rounding algorithm generates the parts. We note that these two added ingredients introduce two main additional obstacles. The first obstacle relates to the last re-balancing step. In both [15, 2] the re-balancing succeeds since it is proved that with a high probability each part by itself is close to being the desired size. The method this is proved is by bounding the variance of the size of each part alone. However, in our approach for Max-3-Section the bound on the variance of the size of a given part depends on the other parts as well. This introduces technical issues and hence bounding of the variance requires much care. The second obstacle relates to the computer assisted proof via *branch and bound* method we employ in order to lower bound the performance ratio of our algorithm. The expression of the separation probability of an edge by the rounding algorithm is involved, as both marginal values are altered when recursing and we employ a random permutation over the order in which the parts are generated. Moreover, a configuration describing how the semi-definite programming relaxation encodes an edge involves 7 different vectors (see Sections 3 and 4). Thus, we had to incorporate many technical ingredients, e.g., analytically bounding the gradient of the separation probability and restricting the search to specific type of configurations while analytically bounding the error this incurs, to make the computer assisted proof terminate faster. This results in about $150,000$ hours of CPU, which is roughly 20 CPU years, to prove Theorem 1.

## 1.2 Additional Related Work

The Max-Bisection problem has a long and rich history. Frieze and Jerrum [6] presented an approximation of 0.6514 based on rounding a semi-definite program. Later on, Ye [16], Halperin and Zwick [10], and Feige and Langberg [5] further improved the approximation guarantee to 0.699, 0.7016, and 0.7027, respectively. They achieved the above by strengthening the semi-definite programming relaxation, e.g., by adding triangle inequality constraints,

and presenting better rounding methods. The next leap in approximating Max-Bisection came with the work of Raghavendra and Tan [15]. They utilized a higher-level Lasserre hierarchy semi-definite program, together with an elegant rounding algorithm, and obtained a 0.85-approximation. Later on, Austrin, Benabbas, and Georgiou [2] showed an improved rounding algorithm, pushing the approximation guarantee up to 0.877. It should be noted that the latter is very close to the best possibloe approximation of 0.878 for Max-Cut.

Focusing on Max-$k$-Cut, Frieze and Jerrum [6] presented an approximation algorithm with better guarantee than the naive random algorithm. They utilized a semi-definite program relaxation alongside an elegant rounding algorithm that samples $k$ random vectors and assigns every vertex $v \in V$ to the cluster of the random vector that is closest to $v$'s vector in the relaxation. Goemans and Williamson [8] presented an improved approximation of 0.836 for Max-3-Cut by using a complex semi-definite program. In [4], de Klerk, Pasechnik, and Warners presented further improved bounds for Max-$k$-Cut. Please refer to Table 1 by Newman [14] for a summary of approximation guarantees.

## 1.3 Paper Organization

We start by presenting preliminary definitions in Section 2. Next, in Section 3 we present our semi-definite program for Max-3-Section and show that it can be strengthened to obtain a solution which is globally uncorrelated. In Section 4 we present our rounding algorithm and its analysis. To obtain a bound on the approximation guarantee of our rounding algorithm, we present an analysis which is based on a computer-assisted proof in Section 5. Furthermore, we discuss the generalization of our algorithm, and its numerical estimation, in Section 6. Missing proofs appear in the full version of the paper.

# 2 Preliminaries

We denote by $\Phi : \mathbb{R} \to [0,1]$ the cumulative distribution function of the normal gaussian distribution and by $\Phi^{-1} : [0,1] \to \mathbb{R}$ its inverse. Specifically, if $R \sim N[0,1]$ then: (1) $\forall x \in \mathbb{R}$: $\Pr[R \leq x] = \Phi(x)$ (or equivalently $\Pr[R \geq x] = 1 - \Phi(x)$); and (2) $\forall x \in [0,1]$: $\Pr[R \leq \Phi^{-1}(x)] = x$ (or equivalently $\Pr[R \geq \Phi^{-1}(x)] = 1 - x$). Moreover, we say that a vector $\mathbf{g}$ is a *random gaussian vector* if its coordinates are i.i.d standard gaussian $N(0,1)$ random variables.

We denote by $\Gamma_t : [0,1]^2 \to [0,1]$ the probability that a standard bi-variate gaussian distribution with correlation $t$ has both its coordinates at most the given quantiles, i.e.,

$$\forall q_1, q_2 \in [0,1] : \ \Gamma_t(q_1, q_2) \triangleq \Pr[X \leq \Phi^{-1}(q_1), Y \leq \Phi^{-1}(q_2)], \ \begin{pmatrix} X \\ Y \end{pmatrix} \sim N\left( \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & t \\ t & 1 \end{pmatrix} \right).$$

We define the mutual information between two random variables.

▶ **Definition 4.** *Let $X, Y$ be jointly distributed random variables taking values in $[q]$. The mutual information of $X$ and $Y$ is defined as*

$$I(X,Y) \triangleq \sum_{i,j \in [q]} \Pr(X = i, Y = j) \log \left( \frac{\Pr(X = i, Y = j)}{\Pr(X = i) \Pr(X = j)} \right).$$

For any two disjoint subsets of vertices $A, B \subseteq V$, we denote by $\delta(A,B)$ the collection of edges having one endpoint in $A$ and another endpoint in $B$. Hence, $|\delta(A,B)|$ denotes the number of edges crossing between $A$ and $B$.[1]

[1] For simplicity of presentation, we assume from this point onward that the graph is unweighted. All of our results apply to graphs equipped with non-negative edge weights, in which case one should substitute $|\delta(A,B)|$ with the total weight of edges crossing between $A$ and $B$.

## 3 The SDP Relaxation for Max-3-Section

In this section, we present a semi-definite programming (SDP) formulation and prove that it is a relaxation for the Max-3-Section problem. Similarly to previous works on Max-Bisection, e.g., [2, 15], we strengthen this formulation and obtain additional properties that will be useful to our rounding algorithm. We define the following SDP formulation for Max-3-Section:

$$\text{(SDP)} \qquad \text{maximize} \quad \sum_{e=(u,v)\in E} \left(1 - \sum_{i=1}^{3} \mathbf{y}_u^i \cdot \mathbf{y}_v^i\right) \tag{1}$$

$$\text{s.t.} \quad \|\mathbf{y}_\emptyset\|^2 = 1 \tag{2}$$

$$\|\mathbf{y}_v^i\|^2 = \mathbf{y}_\emptyset \cdot \mathbf{y}_v^i \qquad \forall v \in V, \forall i = 1,2,3 \tag{3}$$

$$\|\mathbf{y}_v^1\|^2 + \|\mathbf{y}_v^2\|^2 + \|\mathbf{y}_v^3\|^2 = 1 \qquad \forall v \in V \tag{4}$$

$$\mathbf{y}_v^i \cdot \mathbf{y}_v^j = 0 \qquad \forall v \in V, i \neq j \tag{5}$$

$$\mathbf{y}_u^i \cdot \mathbf{y}_v^j \geq 0 \qquad \forall u,v \in V, \forall i,j = 1,2,3 \tag{6}$$

$$\sum_{v\in V} \|\mathbf{y}_v^i\|^2 = n/3 \qquad \forall i = 1,2,3 \tag{7}$$

We note that in the above and what follows, for every vector $\mathbf{y}$ a square norm of a vector $\|\mathbf{y}\|^2$ is with respect to the $\ell_2$ (Euclidean) norm and equals $\mathbf{y} \cdot \mathbf{y}$. Next, we prove that the formulation is a relaxation for our problem. Intuitively, for every vertex $v \in V$ the formulation SDP assigns a distribution over the three clusters via the vectors $\mathbf{y}_v^1$, $\mathbf{y}_v^2$, and $\mathbf{y}_v^3$. Specifically, $\mathbf{y}_\emptyset$ is a unit vectors (Constraint (2)) that denotes *true* whereas the zero vector (that does not appear explicitly in SDP) denotes *false*. Each vector $\mathbf{y}_v^i$ indicates how much vertex $v$ is likely to be assigned to the $i^{\text{th}}$ cluster by SDP. Hence, $\|\mathbf{y}_v^i\|^2$, or equivalently $\mathbf{y}_v^i \cdot \mathbf{y}_\emptyset$ (see Constraint (3)), denotes the marginal probability of assigning vertex $v$ to the $i^{\text{th}}$ cluster by SDP. For every vertex $v \in V$, the sum of these marginal probabilities needs to sum up to one (Constraint 4). Since every vertex $v \in V$ can be assigned to a single cluster in any integral solution, SDP enforces that the vectors $\mathbf{y}_v^i$ and $\mathbf{y}_v^j$ for $i \neq j$ are orthogonal (Constraint (5)). Intuitively, the joint probability SDP assigns for vertices $u$ and $v$ to belong to the $i^{\text{th}}$ and $j^{\text{th}}$ clusters, respectively, is non-negative (Constraint (6)). Finally, since the three clusters are required to be of size $n/3$ each Constraint (7) is added to SDP. When focusing on the objective of SDP (see (1)), for every edge $(u,v) \in E$ the inner product $\mathbf{y}_u^i \cdot \mathbf{y}_v^i$ intuitively indicates the joint probability of both $u$ and $v$ to be assigned to the $i^{\text{th}}$ cluster by SDP. Therefore, intuitively $1 - \mathbf{y}_u^1 \cdot \mathbf{y}_v^1 - \mathbf{y}_u^2 \cdot \mathbf{y}_v^2 - \mathbf{y}_u^3 \cdot \mathbf{y}_v^3$ indicates the likelihood of separating an edge $(u,v)$ by SDP. Thus, this is the objective of SDP.

The following lemma proves that SDP is a relaxation to the Max-3-Section problem, i.e., the value of an optimal solution $\text{OPT}_{\text{SDP}}$ to SDP is an upper bound on the value of an integral optimal solution OPT.

▶ **Lemma 5.** *Given an instance of Max-3-Section let* $\text{OPT}_{\text{SDP}}$ *be the value of an optimal solution of* SDP *(1) and* OPT *be the value of an optimal integral solution. Then,* $\text{OPT}_{\text{SDP}} \geq \text{OPT}$.

**Proof.** Let $\{C_1^*, C_2^*, C_3^*\}$ be an optimal solution for the given instance of Max-3-Section whose value is OPT. Construct the following vector solution to SDP. First, fix an arbitrary unit vector $\mathbf{y}_\emptyset$. Second, for every $v \in V$ define $\mathbf{y}_v^i$ to be the zero vector if $v \notin C_i^*$ and $\mathbf{y}_v^i = \mathbf{y}_\emptyset$ if $v \in C_i^*$. One may notice that all the constraints hold for this solution and the value of the objective of SDP equals the value of the optimal solution $\{C_1^*, C_2^*, C_3^*\}$. Hence, $\text{OPT}_{\text{SDP}} \geq \text{OPT}$. ◀

For simplicity of presentation, we denote by $Y$ a solution to SDP. Thus, $Y$ consists of $\{\mathbf{y}_u^i\}_{u\in V,i=1,2,3}$ and $\mathbf{y}_\emptyset$. A useful property that any feasible solution $Y$ to SDP satisfies is that $\mathbf{y}_u^1+\mathbf{y}_u^2+\mathbf{y}_u^3$ always equals the vector $\mathbf{y}_\emptyset$. This is summarized in the following lemma.

▶ **Lemma 6.** *Let $Y$ be a feasible solution to* SDP*. Then, for every vertex $u \in V$: $\mathbf{y}_u^1+\mathbf{y}_u^2+\mathbf{y}_u^3 = \mathbf{y}_\emptyset$.*

An immediate corollary of the above lemma is that for every pair of vertices $u, v \in V$: $\mathbf{y}_u^i \cdot (\mathbf{y}_v^1+\mathbf{y}_v^2+\mathbf{y}_v^3) = \|\mathbf{y}_u^i\|^2$ (via Constraint (3)).

## 3.1 Globally Uncorrelated Solution

Next we define when a solution $Y$ to SDP is globally uncorrelated following the framework of [15]. Global uncorrelation implies that our rounding algorithm returns a solution that has close to $\frac{n}{3}$ vertices in each part with high probability (see Lemma 15). The sizes then can be corrected with a minor loss in approximation ratio by randomly shifting the imbalanced vertices (see Lemma 16).

A simple fact about any SDP solution is the following: given any two vertices $u, v$, there exists a *local* probability distribution $\mu_{u,v}$ on $\{1,2,3\}^2$ such that $\Pr_{\mu_{u,v}}[X_u = i, X_v = j] = \langle \mathbf{y}_u^i, \mathbf{y}_v^j \rangle$ for every $i, j \in \{1,2,3\}$ and $\Pr_{\mu_{u,v}}[X_u = i] = \|\mathbf{y}_u^i\|^2$ for every $i \in \{1,2,3\}$. The distribution implies that, at least, locally the semi-definite program is a distribution over integral solutions that satisfy correct correlations. The last property states that the distributions are consistent on their intersection which can be at most one vertex.

▶ **Definition 7.** *A solution $Y$ to* SDP *is $\varepsilon$-independent if $\mathbb{E}_{u,v}[I_{\mu_{u,v}}(X_u, X_v)] \leq \varepsilon$ where $\mu_{u,v}$ is the local probability distribution associated with vertices $u$ and $v$.*

The following lemma is an application of Theorem 4.6 from [15] to our SDP for Max-3-Section and it shows how to obtain a $\varepsilon$-independent solution. The algorithm proving Lemma 8 follows from solving the $\Theta(t^2)$-level Lasserre hierarchy semi-definite program for SDP and then inductively conditioning on variables.

▶ **Lemma 8.** *There is an algorithm which, given an integer $t > 0$ and an instance of Max-3-Section, runs in time $n^{O(poly(t))}$ and outputs a set of vectors $Y$ consisting of $\{\mathbf{y}_v^i\}_{v\in V,i=1,2,3}$ and $\mathbf{y}_\emptyset$ such that:*

1. *$Y$ is a feasible solution to* SDP*.*
2. *The objective value of* SDP *(1) when plugging in $Y$ is at least $\mathrm{OPT}_{\mathrm{SDP}} - \frac{1}{t}$.*
3. *$Y$ is $\frac{1}{t}$-independent.*

# 4 The Rounding Algorithm

In this section we present our rounding algorithm for SDP, which appears in Algorithm 1. The algorithm receives as input a solution to SDP and outputs a partition $\{C_1, C_2, C_3\}$ of $V$ that: (1) has high value compared to the SDP value of the input solution; and (2) is (nearly) balanced, i.e., for every $i = 1, 2, 3$: $|C_i|$ is close to $n/3$. We will conclude the analysis by proving that one can re-balance the partition without a significant loss in the value of the solution.

In order to state the algorithm, we require the following definition. For every vertex $u \in V$ and $i = 1, 2, 3$ we denote by $\mathbf{z}_u^i$ the normalized component of $\mathbf{y}_u^i$ that is orthogonal to $\mathbf{y}_\emptyset$, i.e., $\mathbf{y}_u^i = \|\mathbf{y}_u^i\|^2 \mathbf{y}_\emptyset + \sqrt{\|\mathbf{y}_i^i\|^2 - \|\mathbf{y}_u^i\|^4}\mathbf{z}_u^i$. Equivalently,

$$\mathbf{z}_u^i \triangleq \frac{\mathbf{y}_u^i - \|\mathbf{y}_u^i\|^2 \mathbf{y}_\emptyset}{\sqrt{\|\mathbf{y}_u^i\|^2 - \|\mathbf{y}_u^i\|^4}}. \tag{8}$$

Clearly, $\mathbf{z}_u^i$ is a unit vector that is orthogonal to $\mathbf{y}_\emptyset$. We note that if the marginal of vertex $u$ and cluster $i$ is integral, i.e., $\|\mathbf{y}_u^i\|^2 \in \{0, 1\}$, then $\mathbf{z}_u^i$ is not defined. In this case one can simply choose an arbitrary unit vector in the space orthogonal to $\mathbf{y}_\emptyset$ to be $\mathbf{z}_u^i$.

**Algorithm 1** Max-3-Section Rounding Algorithm.

**Input:** solution $\{\mathbf{y}_u^i\}_{u \in V, i=1,2,3}$ and $\mathbf{y}_\emptyset$ to SDP.
**Output:** a partition of $V$ into three parts.

1 Draw uniformly at random a permutation $\pi \in \mathcal{S}_3$.
2 Draw independently two random Gaussian vectors $\mathbf{g}_1$ and $\mathbf{g}_2$.
3 Define the following sets:

$$S_{\pi(1)} \triangleq \left\{ u \in V : \mathbf{z}_u^{\pi(1)} \cdot \mathbf{g}_1 \geq \Phi^{-1}\left(1 - \|\mathbf{y}_u^{\pi(1)}\|^2\right) \right\},$$
$$S_{\pi(2)} \triangleq \left\{ u \in V : \mathbf{z}_u^{\pi(2)} \cdot \mathbf{g}_2 \geq \Phi^{-1}\left(1 - \|\mathbf{y}_u^{\pi(2)}\|^2 / (1 - \|\mathbf{y}_u^{\pi(1)}\|^2)\right) \right\}.$$

4 Return $\{C_1, C_2, C_3\}$ where: $C_{\pi(1)} \triangleq S_{\pi(1)}$, $C_{\pi(2)} \triangleq S_{\pi(2)} \setminus S_{\pi(1)}$, $C_{\pi(3)} \triangleq V \setminus (S_{\pi(1)} \cup S_{\pi(2)})$.

We first prove that Algorithm 1 preserves the marginal probabilities of SDP, i.e., $\|\mathbf{y}_u^i\|^2$ is the probability vertex $u$ is assigned to cluster $C_i$. This is summarized in the following lemma.

▶ **Lemma 9.** *For every $u \in V$ and $i = 1, 2, 3$, it holds that* $\Pr[u \in C_i] = \|\mathbf{y}_u^i\|^2$.

**Proof.** Fix a permutation $\pi \in \mathcal{S}_3$, and let us calculate the probability that $u \in C_i$ conditioned on the event that $\pi$ was chosen in the first step of Algorithm 1. Hence, the following holds for $C_{\pi(1)}$:

$$\Pr\left[u \in C_{\pi(1)} | \pi\right] = \Pr\left[u \in S_{\pi(1)} | \pi\right] = \Pr\left[\mathbf{z}_u^{\pi(1)} \cdot \mathbf{g}_1 \geq \left(1 - \Phi^{-1}\left(1 - \|\mathbf{y}_u^{\pi(1)}\|^2\right)\right) | \pi\right] = \|\mathbf{y}_u^{\pi(1)}\|^2.$$

We observe that the sets $S_{\pi(1)}$ and $S_{\pi(2)}$ are constructed with independent vectors $\mathbf{g}_1$ and $\mathbf{g}_2$. Therefore, similarly to the above, the following holds for $C_{\pi(2)}$:

$$\begin{aligned}
\Pr\left[u \in C_{\pi(2)} | \pi\right] &= \Pr\left[u \notin S_{\pi(1)} \wedge u \in S_{\pi(2)} | \pi\right] \\
&= \Pr\left[u \notin S_{\pi(1)} | \pi\right] \cdot \Pr\left[u \in S_{\pi(2)} | \pi\right] \\
&= \left(1 - \|\mathbf{y}_u^{\pi(1)}\|^2\right) \cdot \Pr\left[\mathbf{z}_u^{\pi(2)} \cdot \mathbf{g}_2 \geq \left(1 - \Phi^{-1}\left(1 - \frac{\|\mathbf{y}_u^{\pi(2)}\|^2}{(1 - \|\mathbf{y}_u^{\pi(1)}\|^2)}\right)\right) \middle| \pi\right] \\
&= \left(1 - \|\mathbf{y}_u^{\pi(1)}\|^2\right) \cdot \frac{\|\mathbf{y}_u^{\pi(2)}\|^2}{(1 - \|\mathbf{y}_u^{\pi(1)}\|^2)} \\
&= \|\mathbf{y}_u^{\pi(2)}\|^2.
\end{aligned}$$

Finally, since the events $\{u \in C_{\pi(1)} | \pi\}$, $\{u \in C_{\pi(2)} | \pi\}$, and $\{u \in C_{\pi(3)} | \pi\}$ are disjoint and exactly one of them occurs, i.e., every vertex $u \in V$ belongs to exactly one cluster in the output, we can conclude that:

$$\Pr\left[u \in C_{\pi(3)} | \pi\right] = 1 - \Pr\left[u \in C_{\pi(1)} | \pi\right] - \Pr\left[u \in C_{\pi(2)} | \pi\right] = 1 - \|\mathbf{y}_u^{\pi(1)}\|^2 - \|\mathbf{y}_u^{\pi(2)}\|^2 = \|\mathbf{y}_u^{\pi(3)}\|^2.$$

In the above the last equality follows from Constraint (4). Unfixing the conditioning on $\pi$ by using the law of total probability concludes the proof. ◀

Our goal is to write an expression for the probability that an edge crosses between two different parts in the partition that Algorithm 1 outputs. Given a fixed pair of vertices $u, v \in V$ and $i = 1, 2, 3$, we denote by $t_i$ the inner product between $\mathbf{z}_u^i$ and $\mathbf{z}_v^i$. One should note that Constraint 3 in SDP 1 implies:

$$t_i = \frac{(\mathbf{y}_u^i - x_i \cdot \mathbf{y}_\emptyset) \cdot (\mathbf{y}_v^i - w_i \cdot \mathbf{y}_\emptyset)}{\sqrt{(x_i - x_i^2) \cdot (w_i - w_i^2)}} = \frac{\alpha_i - x_i w_i}{\sqrt{(x_i - x_i^2) \cdot (w_i - w_i^2)}}, \tag{9}$$

where $x_i \triangleq \|\mathbf{y}_u^i\|^2$ and $w_i \triangleq \|\mathbf{y}_v^i\|^2$ are the marginal values the SDP assigns to vertices $u$ and $v$, respectively, with respect to the $i^{\text{th}}$ cluster, and $\alpha_i \triangleq \mathbf{y}_u^i \cdot \mathbf{y}_v^i$ is the correlation the SDP assigns for both $u$ and $v$ being assigned to the $i^{\text{th}}$ cluster. The following lemma gives the desired expression. We require the following claim for its proof:

▷ Claim 10. Let $(X, Y)$ be a standard bi-variate Gaussian with correlation $t$. Then for every $q_1, q_2 \in [0, 1]$, we have $\Pr[X \geq \Phi^{-1}(1 - q_1), Y \geq \Phi^{-1}(1 - q_2)] = \Gamma_t(q_1, q_2)$.

▶ **Lemma 11.** *For every $u, v \in V$, let $\mathcal{A}_{u,v}$ be the event that Algorithm 1 separates $u$ and $v$:*

$$\Pr\left[\mathcal{A}_{u,v}\right] = 1 - \frac{1}{6} \sum_{\pi \in \mathcal{S}_3} \left[ \Gamma_{t_{\pi(1)}}\left(x_{\pi(1)}, w_{\pi(1)}\right) + \Gamma_{t_{\pi(1)}}\left(1 - x_{\pi(1)}, 1 - w_{\pi(1)}\right) \cdot \right.$$
$$\left. \cdot \left( \Gamma_{t_{\pi(2)}}\left(1 - \frac{x_{\pi(2)}}{1 - x_{\pi(1)}}, 1 - \frac{w_{\pi(2)}}{1 - w_{\pi(1)}}\right) + \Gamma_{t_{\pi(2)}}\left(\frac{x_{\pi(2)}}{1 - x_{\pi(1)}}, \frac{w_{\pi(2)}}{1 - w_{\pi(1)}}\right) \right) \right].$$

The proof of Lemma 11 appears in the full version of the paper.

Our goal now is to lower bound the expected value of the output of Algorithm 1, before it is re-balanced (with a negligible loss) to ensure the size of each cluster is exactly $n/3$. As our analysis is performed edge-by-edge, i.e., for every edge we lower bound the ratio of the probability Algorithm 1 separates the edge to the contribution of this edge to the objective of SDP (1), we introduce the notions of a configuration and a feasible configuration.

A *configuration* is a vector $\mathbf{c} = (x_1, x_2, x_3, w_1, w_2, w_3, \alpha_1, \alpha_2, \alpha_3, t_1, t_2, t_3) \in \mathbb{R}^{12}$, such that for every $i = 1, 2, 3$: $x_i, w_i, \alpha_i \in [0, 1]$ and $t_i \in [-1, 1]$. We say that a configuration $\mathbf{c}$ is a *feasible configuration* if it can be realized by vectors in a feasible solution to SDP (as the following definition states).

▶ **Definition 12.** *A configuration $\mathbf{c} = (x_1, x_2, x_3, w_1, w_2, w_3, \alpha_1, \alpha_2, \alpha_3, t_1, t_2, t_3) \in [0, 1]^9 \times [-1, 1]^3$ is called a* feasible configuration *if there are vectors $\mathbf{y}_u^1$, $\mathbf{y}_u^2$, $\mathbf{y}_u^3$, $\mathbf{y}_v^1$, $\mathbf{y}_v^2$, $\mathbf{y}_v^3$ and $\mathbf{y}_\emptyset$ satisfying:*

1. *The vectors $\mathbf{y}_u^1$, $\mathbf{y}_u^2$, $\mathbf{y}_u^3$, $\mathbf{y}_v^1$, $\mathbf{y}_v^2$, $\mathbf{y}_v^3$ and $\mathbf{y}_\emptyset$ satisfy Constraints (2) to (6) in* SDP.
2. *$x_i = \|\mathbf{y}_u^i\|^2$, $w_i = \|\mathbf{y}_v^i\|^2$ and $\alpha_i = \mathbf{y}_u^i \cdot \mathbf{y}_v^i$, $\forall i = 1, 2, 3$.*
3. *$t_i = (\alpha_i - x_i w_i)/((x_i - x_i^2)(w_i - w_i^2))^{1/2}$, $\forall i = 1, 2, 3$.*

*The vectors $\mathbf{y}_u^1$, $\mathbf{y}_u^2$, $\mathbf{y}_u^3$, $\mathbf{y}_v^1$, $\mathbf{y}_v^2$, $\mathbf{y}_v^3$ and $\mathbf{y}_\emptyset$ are called a* realization *of $\mathbf{c}$. The set of all feasible configuration is denoted by $\mathcal{C}$.*

We note that there is some redundancy in the above definition. First, we can reduce the dimension of the configuration by two simply by substituting $x_3$ with $1-x_1-x_2$ and $w_3$ with $1-w_1-w_2$. Second, we can remove $t_1$, $t_2$ and $t_3$ (or $\alpha_1$, $\alpha_2$ and $\alpha_3$) since each $t_i$ (or $\alpha_i$) can be derived from all the parameters excluding $t_1$, $t_2$, and $t_3$ (or excluding $\alpha_1$, $\alpha_2$, and $\alpha_3$). In Lemma 21, we give a characterization of $\mathcal{C}$ after projecting out $t_1, t_2, t_3$. This description consists of linear constraints and we utilize the description for our computer-assisted proof. For simplicity of the analysis, we will keep all the parameters.

Let us now define two functions over $\mathcal{C}$. The first function $f$, given a feasible configuration $\mathbf{c} \in \mathcal{C}$, returns the probability that Algorithm 1 cuts an edge whose associated vectors are a realization of $\mathbf{c}$. Formally, following Lemma 11:

$$f(\mathbf{c}) \triangleq 1 - \frac{1}{6}\sum_{\pi\in\mathcal{S}_3}\left[\Gamma_{t_{\pi(1)}}\left(x_{\pi(1)}, w_{\pi(1)}\right) + \Gamma_{t_{\pi(1)}}\left(1-x_{\pi(1)}, 1-w_{\pi(1)}\right)\cdot \right.$$
$$\left.\cdot\left(\Gamma_{t_{\pi(2)}}\left(1-\frac{x_{\pi(2)}}{1-x_{\pi(1)}}, 1-\frac{w_{\pi(2)}}{1-w_{\pi(1)}}\right) + \Gamma_{t_{\pi(2)}}\left(\frac{x_{\pi(2)}}{1-x_{\pi(1)}}, \frac{w_{\pi(2)}}{1-w_{\pi(1)}}\right)\right)\right].$$

The second function $g$, given a feasible configuration $\mathbf{c} \in \mathcal{C}$, returns the contribution to the objective of SDP of an edge whose associated vectors are a realization of $\mathbf{c}$. Formally, following (1):

$$g(\mathbf{c}) \triangleq 1 - \alpha_1 - \alpha_2 - \alpha_3.$$

It is important to note that both $f$ and $g$ can be evaluated for every configuration $\mathbf{c} \in [0,1]^9 \times [-1,1]^3$ which might not be necessarily feasible. However, for such a (non feasible) configuration $\mathbf{c}$, $f(\mathbf{c})$ and $g(\mathbf{c})$ lose their "meaning".

To lower bound the value of the solution $\{C_1, C_2, C_3\}$ Algorithm 1 outputs, we introduce the following:

$$\mu \triangleq \inf_{\mathbf{c}\in\mathcal{C}}\left\{\frac{f(\mathbf{c})}{g(\mathbf{c})}\right\}. \tag{10}$$

Clearly, from the above definition of $\mu$, the value of the output $\{C_1, C_2, C_3\}$ of Algorithm 1 is at least: $\mu \cdot \mathrm{OPT}_{\mathrm{SDP}} \geq \mu \cdot \mathrm{OPT}$ (where the inequality follows from Lemma 5 which states that SDP is a relaxation). Hence, if the output $\{C_1, C_2, C_3\}$ of Algorithm 1 was perfectly balanced, i.e., $|C_1| = |C_2| = |C_3| = n/3$, Algorithm 1 would achieve an approximation of $\mu$ to the Max-3-Section problem. In what follows we show that one can re-balance $\{C_1, C_2, C_3\}$ without a significant loss in the value of the solution. Thus, our goal is to lower bound $\mu$. For now, as formally proving the exact value of $\mu$ is a challenging task, we state the following conjecture that follows from numeric estimation of $\mu$.

▶ **Conjecture 13.** $\mu \geq 0.8192$.

Assuming the above conjecture regarding $\mu$ (Conjecture 13), there are two things that are left in order to conclude the analysis of our algorithm. First, we show that if the solution $Y$ to SDP is independent (as in Definition 7 and Lemma 8 ) then with a sufficiently high probability every cluster $C_i$ is close to the desired size of $n/3$. This gives rise to the following Definition 14 and Lemma 15. Second, we show that a solution $\{C_1, C_2, C_3\}$ that is close to being perfectly balanced can be efficiently re-balanced without a significant loss in its value. The latter is summarized in Lemma 16.

▶ **Definition 14.** *A partition* $\{C_1, C_2, C_3\}$ *of a graph on* $n$ *nodes is* $\varepsilon$-unbalanced *if for every* $i = 1, 2, 3$*:*

$$\frac{n}{3}(1-\varepsilon) \leq |C_i| \leq \frac{n}{3}(1+\varepsilon).$$

▶ **Lemma 15.** *Let* $Y$ *be a* $\frac{1}{t}$*-independent solution to* SDP *where* $t = \Omega(\varepsilon^{-18})$*, and* $\{C_1, C_2, C_3\}$ *be the partition that Algorithm 1 outputs on* $Y$*. Then for every* $i = 1, 2, 3$ *it holds that:*

$$Pr\left[||C_i| - n/3| \geq \frac{\varepsilon n}{3}\right] \leq \varepsilon.$$

Next, we show that such unbalanced partition can be balanced without a large loss in the objective. That is, we present an algorithm that given a $\epsilon$-unbalanced partition, finds in polynomial time a balanced partition with small loss in the objective, in expectation.

▶ **Lemma 16.** *There is a polynomial-time algorithm that given a* $\epsilon$*-unbalanced partition* $\{C_1, C_2, C_3\}$ *with value* $\Delta = |\delta(C_1, C_2)| + |\delta(C_2, C_3)| + |\delta(C_1, C_3)|$ *finds a balanced partition* $\{C_1', C_2', C_3'\}$ *with expected value* $\mathbb{E}[\Delta'] \geq (1 - 2\epsilon)\Delta$*.*

The proofs of Lemma 15 and Lemma 16 appear in the full version of the paper. We combine these lemmas and prove the following result.

▶ **Theorem 17.** *For every constant* $\varepsilon > 0$*, there exists a polynomial-time approximation algorithm for* Max-3-Section*, that runs in time* $n^{O(poly(\varepsilon^{-1}))}$*, achieving an approximation of* $(1 - 2\epsilon)(\mu - O(\varepsilon))$*.*

**Proof.** Let $\varepsilon > 0$ be a constant, and $t$ an integer satisfying $t = \Omega(\varepsilon^{-18})$. Lemma 8 shows that we can compute in polynomial time a solution $Y$ to SDP that is $\frac{1}{t}$-independent with only an additive loss of $1/t$ in the objective. We repeatedly apply Algorithm 1 to round the above $Y$ until we obtain a solution $\{C_1, C_2, C_3\}$ that is $\varepsilon$-unbalanced, and then apply Lemma 16 to re-balance it an obtain our final output.

Let us now analyze the approximation guarantee of the above algorithm. First, It follows form Lemma 15 and a simple union bound over the three clusters that with a probability of at least $1 - 3\varepsilon$, applying Algorithm 1 to round the above solution $Y$ yields a clustering $\{C_1, C_2, C_3\}$ that is $\varepsilon$-unbalanced (as in Definition 14). Let us denote by $\mathcal{A}$ the event that $\{C_1, C_2, C_3\}$ is $\varepsilon$-unbalanced. Hence, $\Pr[\mathcal{A}] \geq 1 - 3\varepsilon$ and $\Pr[\bar{\mathcal{A}}] \leq 3\varepsilon$. Moreover, as before, let us denote by $\Delta$ the value of the solution $\{C_1, C_2, C_3\}$. Thus, the expected value of this solution *conditioned* on it being $\varepsilon$-unbalanced is at least:

$$\mathbb{E}[\Delta|\mathcal{A}] = \frac{\mathbb{E}[\Delta] - \Pr[\bar{\mathcal{A}}] \cdot \mathbb{E}[\Delta|\bar{\mathcal{A}}]}{\Pr[\mathcal{A}]} \geq \mu \cdot \mathrm{OPT}_{\mathrm{SDP-H}} - \frac{9\varepsilon}{2}\mathrm{OPT}.$$

The inequality follows from the facts that: (1) $\mathbb{E}[\Delta|\bar{\mathcal{A}}] \leq (3/2) \cdot \mathrm{OPT}$ (since $\Delta \leq m$ and $\mathrm{OPT} \geq (2/3)m$); (2) $\mathbb{E}[\Delta] \geq \mu \cdot \mathrm{OPT}_{\mathrm{SDP-H}}$ (definition of $\mu$ (10)); and (3) $\Pr[\mathcal{A}] \leq 1, \Pr[\bar{\mathcal{A}}] \leq 3\varepsilon$. We note that $\mathrm{OPT}_{\mathrm{SDP-H}} \geq \mathrm{OPT}_{\mathrm{SDP}} - \varepsilon^{18} \geq \mathrm{OPT} - \varepsilon^{18}$. Hence,

$$\mathbb{E}[\Delta|\mathcal{A}] \geq \mathrm{OPT}(\mu - O(\varepsilon)).$$

Applying Lemma 16 concludes the proof. ◀

Moreover, in Observation 34 in the full version of the paper we present a configuration $\mathbf{c}$ that has a ratio of $\frac{f(\mathbf{c})}{g(\mathbf{c})} = 0.8192$, hence that is an upper bound on $\mu$.

## 4.1 Towards Estimating $\mu$ via a Computer Assisted Proof

For our computer assisted proof, we consider a slightly different version of $\mu$ which speeds up our code. Consider the following, for some fixed $\delta' > 0$:

$$\mu' \triangleq \inf_{\mathbf{c}\in\mathcal{C}, g(\mathbf{c})\geq\delta'} \left\{ \frac{f(\mathbf{c})}{g(\mathbf{c})} \right\}. \tag{11}$$

The following lemma shows the loss we incur when using $\mu'$ rather than $\mu$ is bounded.

▶ **Lemma 18.** *Let $\{C_1, C_2, C_3\}$ be the output of Algorithm 1 when run on a solution $Y$ to* SDP *1 with objective value* $\text{SDP}_{\text{VAL}}$. *Then we have that*

$$\mathbb{E}[|\delta(C_1, C_2)| + |\delta(C_1, C_3)| + |\delta(C_2, C_3)|] \geq \left(1 - \frac{\delta'}{2(1-\delta')}\right) \mu' \cdot \text{SDP}_{\text{VAL}}.$$

The following theorem summarizes the approximation guarantee when $\mu'$ is used instead of $\mu$.

▶ **Theorem 19.** *For any constant $\varepsilon > 0$, there is an algorithm that outputs a partition of the vertex set $\{C_1, C_2, C_3\}$ with $|C_1| = |C_2| = |C_3|$ satisfying,*

$$\mathbb{E}[|\delta(C_1, C_2)| + |\delta(C_1, C_3)| + |\delta(C_2, C_3)|] \geq (1-2\epsilon)\left(1 - \frac{\delta'}{2(1-\delta')}\right) \mu'(\text{OPT}_{\text{SDP}} - O(\epsilon^{3/2}))$$

*with an expected run-time of $n^{O(1/\varepsilon)}$.*

**Proof.** First we use Lemma 8 to get a $\frac{1}{t}$-independent solution $Y$ for $t = \Omega(\epsilon^{-18})$. Let $\text{OPT}_{\text{SDP}-\text{H}}$ denote the objective value of this solution. Next we run Algorithm 1 on $Y$ and repeat until it outputs sets $C_1', C_2', C_3'$ that are $\epsilon$-unbalanced. We note that by Lemma 15 $(C_1', C_2', C_3')$ is *not* $\epsilon$-unbalanced with probability at most $3\epsilon$, so we run Algorithm 1 at most $1/3\epsilon$ times in expectation. Since $C_1', C_2', C_3'$ are $\epsilon$-unbalanced, applying the random shifting of Lemma 16 to $C_1', C_2', C_3'$ we get $C_1, C_2, C_3$ satisfying

$$\mathbb{E}[|\delta(C_1, C_2)| + |\delta(C_1, C_3)| + |\delta(C_2, C_3)|] \geq (1-2\epsilon)\mathbb{E}[|\delta(C_1', C_2')| + |\delta(C_1', C_3')| + |\delta(C_2', C_3')|],$$

From Lemma 18 we have,

$$\mathbb{E}[|\delta(C_1, C_2)| + |\delta(C_1, C_3)| + |\delta(C_2, C_3)|] \geq (1-2\epsilon)\left(1 - \frac{\delta'}{2(1-\delta')}\right) \mu' \text{OPT}_{\text{SDP}-\text{H}}.$$

Then applying second item of Lemma 8 gives us that $\text{OPT}_{\text{SDP}-\text{H}} \geq \text{OPT}_{\text{SDP}} - O(\epsilon^{3/2})$ since we set $t = \Omega(\epsilon^{-18})$. ◀

# 5 Computer Assisted Proof

The goal of this section is to lower bound $\mu'$. We use a branch and bound procedure to lower $\mu'$ and which gives a guarantee for the approximation factor of our algorithm (Theorem 19) . We recall the definition of $\mu'$

$$\mu' \triangleq \inf_{\mathbf{c}\in\mathcal{C}, g(\mathbf{c})\geq\delta'} \left\{ \frac{f(\mathbf{c})}{g(\mathbf{c})} \right\}. \tag{12}$$

The following claim gives a simple, yet useful, bound on the probability that our rounding algorithm will separate two vertices in the graph.

▷ Claim 20. Let $u, v \in V$, let $x_i = \|\mathbf{y}_u^i\|^2$ and $w_i = \|\mathbf{y}_v^i\|^2$ for $i = 1, 2, 3$ and $\mathbf{c}$ be a configuration corresponding to this pair. Then it holds that

$$f(\mathbf{c}) \geq \frac{|x_1 - w_1| + |x_2 - w_2| + |x_3 - w_3|}{2}.$$

We characterize the configuration space $\mathcal{C}$ which we will consider in the computer assisted proof.

▶ **Lemma 21.** *Let* $\mathbf{c} = (x_1, x_2, x_3, w_1, w_2, w_3, \alpha_1, \alpha_2, \alpha_3, t_1, t_2, t_3) \in \mathcal{C}$. *Then* $\mathbf{c}$ *satisfies the following:*

1. $x_1 + x_2 + x_3 = w_1 + w_2 + w_3 = 1$.
2. $0 \leq \alpha_i \leq \min\{x_i, w_i\}$ *for all* $i = 1, 2, 3$.
3. $\max\{0, x_3 - \alpha_3 + \alpha_1 - w_1, w_2 - \alpha_2 + \alpha_1 - x_1\} \leq \min\{w_2 - \alpha_2, x_3 - \alpha_3, x_2 + x_3 - w_1 + \alpha_1 - \alpha_2 - \alpha_3\}$.
4. $t_i = \frac{\alpha_i - x_i w_i}{\sqrt{(x_i - x_i^2)(w_i - w_i^2)}}$ *for* $i \in [3]$.

We define the following two polytopes which we will consider when verifying bounding $\mu'$. These polytopes help us to speed up the branch and bound procedure.

1. $\mathcal{S} \triangleq \{(x_1, x_2, x_3, w_1, w_2, w_3, \alpha_1, \alpha_2, \alpha_3, t_1, t_2, t_3) | x_1 \leq \min(x_2, w_1, w_2, w_3, x_3), x_2 \leq x_3\}$.
2. $\mathcal{E} \triangleq \{\mathbf{c} = (x_1, x_2, x_3, w_1, w_2, w_3, \alpha_1, \alpha_2, \alpha_3, t_1, t_2, t_3)\}$ such that

$$\sum_{i=1}^{3} \frac{|x_i - w_i|}{2} \leq \rho g(\mathbf{c}), g(\mathbf{c}) \geq \delta'\}.$$

The following claim shows why we can restrict to configurations in $\mathcal{S}$ and follows from the symmetry of $f, g$.

▷ Claim 22. Let $\mathbf{c} \in \mathcal{C} - \mathcal{S}$. Then there exists $\mathbf{c}' \in \mathcal{C} \cap \mathcal{S}$ such that $f(\mathbf{c}) = f(\mathbf{c}')$ and $g(\mathbf{c}) = g(\mathbf{c}')$.

Then Claim 22 and Claim 20 imply the following.

▷ Claim 23. If $\inf_{\mathbf{c} \in \mathcal{C} \cap \mathcal{S} \cap \mathcal{E}} \frac{f(\mathbf{c})}{g(\mathbf{c})} \geq \rho$ then $\mu' \geq \rho$.

Proof. By Claim 22 we get the following: $\mu' = \inf_{\mathbf{c} \in \mathcal{C}, g(\mathbf{c}) \geq \delta'} \frac{f(\mathbf{c})}{g(\mathbf{c})} = \inf_{\mathbf{c} \in \mathcal{C} \cap \mathcal{S}, g(\mathbf{c}) \geq \delta'} \frac{f(\mathbf{c})}{g(\mathbf{c})}$. Assume for contradiction that $\mu' = \min_{\mathbf{c} \in \mathcal{C} \cap \mathcal{S}, g(\mathbf{c}) \geq \delta'} \frac{f(\mathbf{c})}{g(\mathbf{c})} < \rho$. Then $\exists \mathbf{c}' \in \mathcal{C} \cap \mathcal{S}$ with $g(\mathbf{c}') \geq \delta'$ and $\frac{f(\mathbf{c}')}{g(\mathbf{c}')} < \rho$. We have that $\mathbf{c}' \notin \mathcal{E}$ otherwise $g(\mathbf{c}') \geq \rho$ by the assumption of the lemma. Then by definition of $\mathcal{E}$, $\sum_{i=1}^{3} \frac{|x_i - w_i|}{2} > \rho g(\mathbf{c}')$, but this is a contradiction by Claim 20 since $f(\mathbf{c}') > \sum_{i=1}^{3} \frac{|x_i - w_i|}{2} > \rho g(\mathbf{c}')$. ◁

Thus our goal for the computed assisted proof is to show $\inf_{\mathbf{c} \in \mathcal{C} \cap \mathcal{S} \cap \mathcal{E}} \frac{f(\mathbf{c})}{g(\mathbf{c})} \geq \rho$ for some value of $\rho$. Given 12 intervals $(I_1, \ldots, I_{12})$ we define the following polytopes to divide our feasible region into hypercubes. Consider the following polytope,

$$\mathcal{P}(I_1, \ldots, I_{12}) = (I_1 \times I_2 \ldots \times I_{12}) \cap \mathcal{C}.$$

Note that intervals $I_j$ correspond to possible values of $x_j$ for $j \in [3]$, intervals $I_3, I_4, I_5$ correspond to values of $w_1, w_2, w_3$, intervals $I_7, I_8, I_9$ correspond to values of $\alpha_1, \alpha_2, \alpha_3$, and intervals $I_{10}, I_{11}, I_{12}$ correspond to $t_1, t_2, t_3$. Our computer assisted proof enumerates $I_1, \ldots, I_{12}$ so that the union of all $\mathcal{P}(I_1, \ldots, I_{12}) \cap \mathcal{S} \cap \mathcal{E}$ covers $\mathcal{C} \cap \mathcal{S} \cap \mathcal{E}$. For each $I_1, \ldots, I_{12}$ we show one of the following three:

1. $\mathcal{P}(I_1, \ldots, I_{12}) \cap \mathcal{S} \cap \mathcal{E} = \emptyset$.
2. $\inf_{\mathbf{c} \in \mathcal{P}(I_1,\ldots,I_{12}) \cap \mathcal{S} \cap \mathcal{E}} \frac{f(\mathbf{c})}{g(\mathbf{c})} \geq \rho$.
3. Divide $(I_1, \ldots, I_{12})$ into a collection $\mathcal{U}$ whose union equals $(I_1, \ldots, I_{12})$ so that each $(I'_1, \ldots, I'_{12}) \in \mathcal{U}$ satisfies one of the first two items.

which implies the hypothesis of Claim 23. The third item is the branching step and the first two items are how we eliminate branches. For our computer assisted proof we will only consider $x_1, x_2, w_1, w_2, t_1, t_2, t_3$ as independent variables. The remaining variables $w_3, x_3, \alpha_1, \alpha_2, \alpha_3$ will always take values $w_3 = 1 - w_1 - w_2, x_3 = 1 - x_1 - x_2, \alpha_i = x_i w_i + t_i \sqrt{(x_i - x_i^2)(w_i - w_i^2)}$ for $i \in [3]$. Our algorithm runs in stages. In the first stage we use an LP to eliminate hypercubes. The first stage works as follows,

1. Enumerate all $\mathcal{I} = (I_1, \ldots, I_{12})$ such that $|I_j| = \eta_1$ for $j \in \{1, 2, 4, 5\}$ and $|I_j| = \eta_2$ for $j = 10, 11, 12$ that the union of all $(I_1, I_2, I_3, I_4, I_{10}, I_{11}, I_{12})$ covers the region $[0, 1]^4 \times [-1, 1]^3$. Note that intervals $I_5, I_6, I_7$ can be determined by the bounds on the other intervals, $I_j$ such that $j \notin \{5, 6, 7\}$. This follows since bounds on $x_i, w_i, t_i$ imply bounds on $\alpha_i$ because $\alpha_i = t_i \sqrt{(x_i - x_i^2)(w_i - w_i^2)} + x_i w_i$. Similarly, $I_3, I_6$ are determined by $I_1, I_2$ and $I_4, I_5$ respectively since $x_3 = 1 - x_1 - x_2$ and $w_3 = 1 - w_1 - w_2$.
2. For each hypercube $\mathcal{I}$ enumerated in the previous step, check that $\mathcal{P}(\mathcal{I}) \cap \mathcal{E} \cap \mathcal{S}$ contains a feasible point by solving an LP. If yes, save $\mathcal{I}$ for further processing.

### Partial Derivatives

A crucial ingredient of the branch and bound procedure is to obtain a lower bound on the function $f(\mathbf{c})$ for any configuration $\mathbf{c} \in \mathcal{I}$ in any cube. This we do by computing $f(\mathbf{c}^*)$ for some well chosen $\mathbf{c}^* \in \mathcal{I}$ and then using bounds on the partial derivatives to infer a bound $f(\mathbf{c})$ for all other $\mathbf{c} \in \mathcal{I}$. A tight bound on partial derivatives ensures a smaller branch and bound tree. We detail the partial derivatives and bounds thus obtained now. Previously, we defined the notion of configuration and the function $f$ as a function of 12 variables. Since we only consider $(x_1, x_2, w_1, w_2, t_1, t_2, t_3)$ as variables we have $\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial \alpha_i} = 0$ and $f, g$ are functions of 7 independent variables. We recall the definition of $f$

$$f(x, w, \alpha, t) = 1 - \frac{1}{6} \sum_{\pi \in \mathcal{S}_3} \Bigg[ \Gamma_{t_{\pi(1)}} \left( x_{\pi(1)}, w_{\pi(1)} \right) + \Gamma_{t_{\pi(1)}} \left( 1 - x_{\pi(1)}, 1 - w_{\pi(1)} \right) \cdot \\ \cdot \left( \Gamma_{t_{\pi(2)}} \left( 1 - \frac{x_{\pi(2)}}{1 - x_{\pi(1)}}, 1 - \frac{w_{\pi(2)}}{1 - w_{\pi(1)}} \right) + \Gamma_{t_{\pi(2)}} \left( \frac{x_{\pi(2)}}{1 - x_{\pi(1)}}, \frac{w_{\pi(2)}}{1 - w_{\pi(1)}} \right) \right) \Bigg].$$

Moreover, for our use we can assume that the domain of $f$ is $0 \leq x_1 + x_2 \leq 1$, $0 \leq w_1 + w_2 \leq 1$, $\alpha_1, \alpha_2, \alpha_3 \in [0, 1]$ and $t_1, t_2, t_3 \in [-1, 1]$. In the following, we are going to prove bounds on the partial derivatives of $f(x, w, \alpha, t)$ with respect to $x_i$, $w_i$ and $t_i$.

▶ **Lemma 24.** *For each $(x, w, \alpha, t)$ in the domain of configuration, the following bounds on the partial derivatives of $f$ hold:*

1. $\frac{\partial}{\partial t_i} f(x, w, \alpha, t) < 0$, *for* $i = 1, 2, 3$.
2. $|\frac{\partial}{\partial x_i} f(x, w, \alpha, t)| \leq \frac{5}{3} - \frac{1}{3}(1 - x_{3-i})$, *for* $i = 1, 2$.
3. $|\frac{\partial}{\partial w_i} f(x, w, \alpha, t)| \leq \frac{5}{3} - \frac{1}{3}(1 - w_{3-i})$, *for* $i = 1, 2$.

The proof of the lemma is technical and appears in the full version of the paper. We now have the following claim that gives a lower bound on all configurations in the hypercube as compared to the point $\mathbf{m}$ in it.

▷ Claim 25. Let $\mathcal{I}$ be a hypercube and $\mathcal{Q} \triangleq \mathcal{P}(\mathcal{I}) \cap \mathcal{S} \cap \mathcal{E}$. Let $\mathbf{m} \in \mathbb{R}^{12}$ be a point where the coordinates corresponding to $x_1, x_2, w_1, w_2$ are the midpoints $I_1, I_2, I_4, I_5$ respectively and the last 3 coordinates corresponding are $\overline{t_1}, \overline{t_2}, \overline{t_3}$ which are the upper bounds of $I_{10}, I_{11}, I_{12}$. Then the following holds,

$$\min_{\mathbf{c}=(x_1,\ldots,t_1,t_2,t_3)\in\mathcal{Q}} f(x_1, x_2, \ldots, \overline{t_1}, \overline{t_2}, \overline{t_3}) \geq f(\mathbf{m}) - \frac{1}{6} \sum_{z\in\{x,w\}} \sum_{i=1}^{2} (5 - (1 - \overline{z_{3-i}}))(\overline{z_i} - \underline{z_i}).$$

▶ **Lemma 26.** *Let $\mathcal{I}, \mathbf{m}, \mathcal{Q}$ be defined as in Claim 25. Then $\min_{\mathbf{c}\in\mathcal{Q}} \frac{f(\mathbf{c})}{g(\mathbf{c})} \geq \rho$ if,*

$$f(\mathbf{m}) - \frac{1}{6} \sum_{z\in\{x,w\}} \sum_{i=1}^{2} (5 - (1 - \overline{z_{3-i}}))(\overline{z_i} - \underline{z_i}) \geq \rho \max_{\mathbf{c}\in\mathcal{Q}} g(\mathbf{c})$$

*where $\overline{z_i}, \underline{z_i}$ for $z \in \{w, x\}$ are the upper and lower bounds given by their corresponding interval in $\mathcal{I}$.*

**Proof.** We use the fact that $f$ is non-increasing in $t_i$ and Claim 25 to get,

$$\begin{aligned}\min_{\mathbf{c}\in\mathcal{Q}} f(\mathbf{c}) &\geq \min_{\mathbf{c}=(x_1,x_2,\ldots,t_1,t_2,t_3)\in\mathcal{Q}} f(x_1, x_2, \ldots, \overline{t_1}, \overline{t_2}, \overline{t_3}) \\ &\geq f(\mathbf{m}) - \frac{1}{6} \sum_{z\in\{x,w\}} \sum_{i=1}^{2} (5 - (1 - \overline{z_{3-i}}))(\overline{z_i} - \underline{z_i})\end{aligned}$$

The first inequality follows since $f$ is non-increasing in $t, \alpha$ by Lemma 24. The third inequality follows by Claim 25. Thus by the inequality in the lemma and the relation above, we have shown: $\min_{\mathbf{c}\in\mathcal{Q}} f(\mathbf{c}) \geq \rho \max_{\mathbf{c}\in\mathcal{Q}} g(\mathbf{c})$. ◀

Now we describe the final stage of the experiment which eliminates all remaining cubes from the first stage.

1. For all remaining hyper cubes $\mathcal{I}$ from stage 1 run the following steps until all cubes are eliminated.
2. Split the intervals $I_1, I_2, I_4, I_5$ corresponding to $x_1, x_2, w_1, w_2$ into halves to get 16 smaller sub-hypercubes $(I'_1, \ldots, I'_9, I_{10}, I_{11}, I_{12})$. We note that this also tightens the intervals corresponding to $x_3, w_3, \alpha_1, \alpha_2, \alpha_3$. Check if each smaller hypercube has a feasible point in $\mathcal{P}(I'_1, \ldots, I'_9, I_{10}, I_{11}, I_{12}) \cap \mathcal{S} \cap \mathcal{E}$. If yes, then the sub-hypercube can be eliminated.
3. Otherwise, verify if the inequality in 26 holds. If yes, the sub-hypercube can be eliminated.
4. Otherwise, split the $t_1, t_2, t_3$ intervals into halves to get 8 sub-hypercubes $\mathcal{I}' = (I'_1, \ldots, I'_9, I''_{10}, I''_{11}, I''_{12})$. This also tightens the intervals for $\alpha_1, \alpha_2, \alpha_3$. Check if each smaller hypercube $\mathcal{I}'$ has a feasible point in $\mathcal{P}(\mathcal{I}') \cap \mathcal{S} \cap \mathcal{E}$. If yes, then the sub-hypercube can be eliminated.
5. Otherwise, split the sub-hyper cube into 16 smaller hyper-cubes using Step 2. Repeat Steps 2-5 until all sub cubes are eliminated.

We ran this branch and bound procedure with $\rho = 0.80, \delta' = 0.01$ giving a final approximation of 0.795.

## 6 Max-k-Section

Our algorithm for Max-3-Section can be generalized for larger number of sections in the following way. For a natural number $k \geq 4$, one can write a similar SDP formulation, where each node $v \in V$ has $k$ vectors $\mathbf{y}_v^1, \ldots, \mathbf{y}_v^k$, with similar constraints and objective for the SDP

for Max-3-Section: maximize $\sum_{\{u,v\}\in E}\cdot(1-\sum_{i=1}^{k}\mathbf{y}_u^i\cdot\mathbf{y}_v^i)$. The rounding algorithm will work in a similar way. For example, we consider $k=4$. We draw a random permutation $\pi$ in $\mathcal{S}^4$ and three random gaussian vectors $\mathbf{g}_1,\mathbf{g}_2,\mathbf{g}_3$ with coordinates independently distributed by $N(0,1)$. Then, we define $S_{\pi(1)}$ and $S_{\pi(2)}$ in the same way like in the algorithm for Max-3-Section. Next, we define

$$S_{\pi(3)}\triangleq\left\{u\in V:\mathbf{z}_u^{\pi(3)}\cdot\mathbf{g}_3\geq\Phi^{-1}\left(1-\frac{\|\mathbf{y}_u^{\pi(3)}\|^2}{1-\|\mathbf{y}_u^{\pi(1)}\|^2-\|\mathbf{y}_u^{\pi(2)}\|^2}\right)\right\}$$

and the four clusters in the output will be $C_{\pi(1)}=S_{\pi(1)}$, $C_{\pi(2)}=S_{\pi(2)}\setminus S_{\pi(1)}$, $C_{\pi(3)}=S_{\pi(3)}\setminus(S_{\pi(2)}\cup S_{\pi(1)})$ and $C_{\pi(4)}=V\setminus(S_{\pi(1)}\cup S_{\pi(2)}\cup S_{\pi(3)})$. Then, for any constant $k$, we can claim that with high probability the solution is concentrated and can be re-balanced, similarly to Lemma 15 and Lemma 16.

Our goal now is to bound the approximation factors that these algorithms achieve, for each $k$. As we discussed in the previous sections, computing the worst approximation guarantee, or even bounding it analytically is not an easy task. For $k=3$, we used the branch and bound algorithm and presented a lower bound on the approximation ratio. For larger values of $k$, the computer-assisted proof method becomes computationally harder. However, one possible approach is to try and give a numerical estimation for the approximation factor. We will do that in the following way: for each $k$, one can write an optimization problem over $k^2$ variables that represent the inner products $\mathbf{y}_u^i\cdot\mathbf{y}_v^j$, for each $i,j\in[k]$, or as we denoted before, a feasible configuration. Then, we wish to minimize the ratio of the probability that $u,v$ are separated and the contribution of the edge $\{u,v\}$ to the SDP, which is $1-\sum_{i=1}^{k}\mathbf{y}_u^i\cdot\mathbf{y}_v^i$. To solve that optimization problem, we use Matlab and the *fmincon* functionality which can find a local minimum for this optimization problem, but only a local minimum. Therefore, we repeat the experiment for numerous random starting points in the feasible region. The results of the numerical estimations of the approximation for $k=3,4,5$ are presented in Conjecture 3.

We note that given a configuration of vectors $\mathbf{y}_u^1,\mathbf{y}_u^2,\mathbf{y}_u^3,\mathbf{y}_v^1,\mathbf{y}_v^2,\mathbf{y}_v^3$ that has a ratio of $\rho$ between the separation probability and the contribution of the edge $(u,v)$ to the SDP solution for Max-3-section, one can construct a configuration for max-4-section by adding two zero vectors $\mathbf{y}_u^4,\mathbf{y}_v^4$. That configuration will have the same contribution for the SDP for max-4-section, and the separation probability will also be the same, even though the algorithm admits four sections. Therefore, in that way of analysis, the approximation ratio of our algorithm can only decrease as $k$ increases. However, our numerical estimations show that for $k\leq 5$, the approximation is not worse than 0.8192. In addition, we note that the simple algorithm that returns a random balanced $k$-partition achieves a $1-\frac{1}{k}$ approximation, hence for $k=3,4,5$ our algorithm surpasses it.

## References


1 Gunnar Andersson. An approximation algorithm for max p-section. In *STACS 99: 16th Annual Symposium on Theoretical Aspects of Computer Science Trier, Germany, March 4–6, 1999 Proceedings*, pages 237–247. Springer, 2002.

2 Per Austrin, Siavosh Benabbas, and Konstantinos Georgiou. Better balance by being biased: A 0.8776-approximation for max bisection. *ACM Trans. Algorithms*, 13(1):2:1–2:27, 2016.

3 Etienne De Klerk, Dmitrii Pasechnik, Renata Sotirov, and Cristian Dobre. On semidefinite programming relaxations of maximum k-section. *Mathematical programming*, 136(2):253–278, 2012.

**4** Etienne de Klerk, Dmitrii V Pasechnik, and Joost P Warners. On approximate graph colouring and max-k-cut algorithms based on the $\theta$-function. *Journal of Combinatorial Optimization*, 8:267–294, 2004.

**5** Uriel Feige and Michael Langberg. The rpr2 rounding technique for semidefinite programs. *Journal of Algorithms*, 60(1):1–23, 2006.

**6** Alan Frieze and Mark Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18(1):67–81, 1997.

**7** Daya Ram Gaur, Ramesh Krishnamurti, and Rajeev Kohli. The capacitated max k-cut problem. *Mathematical Programming*, 115:65–72, 2008.

**8** Michel X Goemans and David Williamson. Approximation algorithms for max-3-cut and other problems via complex semidefinite programming. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 443–452, 2001.

**9** Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

**10** Eran Halperin and Uri Zwick. A unified framework for obtaining improved approximation algorithms for maximum graph bisection problems. *Random Structures & Algorithms*, 20(3):382–402, 2002.

**11** Richard M Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 1972.

**12** Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, 2007.

**13** Ai-fan Ling. Approximation algorithms for max 3-section using complex semidefinite programming relaxation. In *Combinatorial Optimization and Applications: Third International Conference, COCOA 2009, Huangshan, China, June 10-12, 2009. Proceedings 3*, pages 219–230. Springer, 2009.

**14** Alantha Newman. Complex semidefinite programming and max-k-cut. In *SIAM Symposium on Simplicity in Algorithms*, 2018.

**15** Prasad Raghavendra and Ning Tan. Approximating csps with global cardinality constraints using SDP hierarchies. In *SODA*, pages 373–387. SIAM, 2012.

**16** Yinyu Ye. A .699-approximation algorithm for max-bisection. *Mathematical Programming*, 90(1):101–111, March 2001.

# Fitting Tree Metrics with Minimum Disagreements

**Evangelos Kipouridis**
Saarland University, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarbrücken, Germany

## Abstract

In the $L_0$ Fitting Tree Metrics problem, we are given all pairwise distances among the elements of a set $V$ and our output is a tree metric on $V$. The goal is to minimize the number of pairwise distance disagreements between the input and the output. We provide an $O(1)$ approximation for $L_0$ Fitting Tree Metrics, which is asymptotically optimal as the problem is APX-Hard.

For $p \geq 1$, solutions to the related $L_p$ Fitting Tree Metrics have typically used a reduction to $L_p$ Fitting Constrained Ultrametrics. Even though in FOCS '22 Cohen-Addad et al. solved $L_0$ Fitting (unconstrained) Ultrametrics within a constant approximation factor, their results did not extend to tree metrics.

We identify two possible reasons, and provide simple techniques to circumvent them. Our framework does not modify the algorithm from Cohen-Addad et al. It rather extends any $\rho$ approximation for $L_0$ Fitting Ultrametrics to a $6\rho$ approximation for $L_0$ Fitting Tree Metrics in a blackbox fashion.

**2012 ACM Subject Classification** Theory of computation → Facility location and clustering; Theory of computation → Random projections and metric embeddings

**Keywords and phrases** Hierarchical Clustering, Tree Metrics, Minimum Disagreements

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.70

**Related Version** *Full Version*: `https://arxiv.org/abs/2307.16066`

**Funding** This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

## 1 Introduction

Trees are used by many disciplines to describe relationships between entities. For example, in biology, the universal tree of life describes evolutionary distances between organisms. In fact, trees are relevant for any historical science studying an evolutionary branching process (e.g. historical linguistics and sociocultural evolution).

In these cases, we are guaranteed that the underlying truth can be described by a tree. This underlying tree may even have a special structure. For example in machine learning and data analysis (see e.g.: [3]) it may be an ultrametric, that is a rooted tree with all leaves being at the same depth. In any case, our access to this tree is usually only through estimations of pairwise distances. A natural task is thus the reconstruction of the tree, given (noisy) measurements of pairwise distances.

As the noisy measurements may not describe a tree, we are interested in finding the "closest" tree to the input. In this work we study the problem of minimizing the number of pairwise distance disagreements between the measurements and the output tree. As noted in [6], this objective has a practical relevance; often the distances are obtained by different (human) classifiers. It is expected that most will do a good job, but if an error occurs, it may be by a large amount.

Other objectives have also been studied, e.g. minimizing the total error [2, 5, 8], or minimizing the maximum error [1]. In order to formally introduce a class of problems that captures all aforementioned objectives we first make some definitions.


## 1.1 Problem Definitions

Given a set $V$, we denote by $\binom{V}{2}$ the set of all (unordered) pairs of disjoint elements from set $V$. We use the term distance matrix to refer to a function from $\binom{V}{2}$ to the non-negative reals. Let $D$ be a distance matrix. We slightly abuse notation and say that for any $u \in V$, $D(u,u) = 0$. For $p \geq 1$, we say that $\|D\|_p = \sqrt[p]{\sum_{\{u,v\} \in \binom{V}{2}} |D(u,v)|^p}$ is the $L_p$ norm of $D$. We extend the notation for $p = 0$. In this case $\|D\|_0$ denotes the number of pairs $\{u,v\}$ such that $D(u,v) \neq 0$. We even say $\|D\|_0$ is the $L_0$ norm of $D$, despite $L_0$ not being a norm. For ease of notation, we use $0^0 = 0$, so that $x^0 = 0$ if $x = 0$, and 1 otherwise. As in [6] (and implicitly in [1]), we allow tree metrics and ultrametrics to have distances equal to 0.

▶ **Definition 1.** *In the $L_p$ Fitting Tree (Ultra) Metrics problem, we are given as input a set $V$ and a distance matrix $D$.*

*The output is a tree metric (or ultrametric) $T$ that spans $V$ and fits $D$ in the sense of minimizing the $L_p$-norm*

$$\|T - D\|_p = \sqrt[p]{\sum_{\{u,v\} \in \binom{V}{2}} |T(u,v) - D(u,v)|^p}.$$

We also define a similar problem, $L_p$ Fitting Constrained Ultrametrics. It was initially defined in [1], who proved that, for $p \geq 1$, a $\rho$ approximation for $L_p$ Fitting Constrained Ultrametrics translates to a $3\rho$ approximation for $L_p$ Fitting Tree Metrics.

▶ **Definition 2.** *In the $L_p$ Fitting Constrained Ultrametrics problem, we are given as input a set $V$, a distance matrix $D$, a distinguished element $\alpha \in V$, a positive number $h$ and a positive number $l_u$ for each $u \in V$. In particular it holds that $l_\alpha = h$.*

*The output is an ultrametric $U$ that spans $V$. It shall also hold that*

$$\max\{l_u, l_v\} \leq U(u,v) \leq h \qquad \forall \{u,v\} \in \binom{V}{2}.$$

*$U$ shall fit $D$ in the sense of minimizing the $L_p$-norm*

$$\|U - D\|_p = \sqrt[p]{\sum_{\{u,v\} \in \binom{V}{2}} |U(u,v) - D(u,v)|^p}.$$

## 1.2 Previous work

When the input is a tree metric, a corresponding tree can be found in $O(|V|^2)$ time (linear in the input size) [9]. As this is usually not the case, research focused on $L_p$ Fitting Tree Metrics.

The first $L_p$ Fitting Tree Metrics problem solved within an asymptotically optimal approximation factor is the $L_\infty$ Fitting Tree Metrics problem, by Agarwala et al. [1]. In order to solve it, the authors give a reduction to the $L_\infty$ Fitting Constrained Ultrametrics problem which increases the approximation by a factor 3. They then use the exact solution of this problem from [7]. In the same paper, they also show how to extend this reduction for any $L_p$ norm, $p \geq 1$.

This reduction turned out to be an essential tool for tackling $L_p$ Fitting Tree Metrics. Harp, Kannan and McGregor [8] developed an $O(\min\{n, k \log n\}^{1/p})$ approximation factor for $L_p$ Fitting Ultrametrics, $p \geq 1$, where $k$ is the number of distinct distances in the input. Using

the reduction from [1], they extend their result to the $L_p$ Fitting Tree Metrics case[1]. Similarly, Ailon and Charikar [2] get an $O(((\log n)(\log\log n))^{1/p})$ approximation for the ultrametrics case, which they then extend to the tree metrics case using the well established reduction. Finally, Cohen-Addad et al. [5] achieve an asymptotically optimal $O(1)$ approximation factor for $L_1$ Fitting Tree Metrics, again using an asymptotically optimal $O(1)$ approximation factor for the ultrametrics case.

In FOCS '22 Cohen-Addad et al. [6] solved the $L_0$ Fitting Ultrametrics problem within an asymptotically optimal $O(1)$ approximation factor. However, their result was not extended to the $L_0$ Fitting Tree Metrics problem. We identify two possible reasons for that:

- Most importantly, the reduction from [1] does not work for $L_0$. The reason is that a crucial step of it uses the convexity of all $L_p$-norms, $p \geq 1$. $L_0$ however is not convex (and in fact is not a norm).
- Even if the reduction worked for $L_0$, the algorithm for $L_0$ Fitting Ultrametrics should be extended to the $L_0$ Fitting Constrained Ultrametrics problem.

## 1.3 Our results

In this work we show how any $\rho$ approximation for $L_0$ Fitting Ultrametrics can be extended to a $6\rho$ approximation for $L_0$ Fitting Tree Metrics.

In particular, we extend the reduction from [1] to the $L_0$ case, despite $L_0$ not being convex. We do so by avoiding the averaging argument from [1] which required convexity, and was necessary to prove the existence of a node with certain properties. Our argument is of course only valid for $L_0$.

Furthermore, we show how one can use any algorithm for $L_0$ Fitting Ultrametrics to solve $L_0$ Fitting Constrained Ultrametrics, in a blackbox manner. In contrast [1, 2, 8] all needed to apply ad-hoc modifications to their $L_p$ Fitting Ultrametrics algorithms to also solve $L_p$ Fitting Constrained Ultrametrics.

An immediate corollary of these two results is that the ultrametrics algorithm from [6] can be used to get an asymptotically optimal $O(1)$ approximation factor for $L_0$ Fitting Tree Metrics. Even though this constant is large, any improved approximation factor for $L_0$ Fitting Ultrametrics would immediately yield an improved approximation for $L_0$ Fitting Tree Metrics, using this framework.

Finally, we prove that $L_0$ Fitting Tree Metrics is APX-Hard.

It is interesting to notice that apart from avoiding the averaging argument from [1], the rest follow existing techniques. However, due to the special structure of $L_0$, we significantly simplify them.

# 2 From tree metrics to ultrametrics

In this section we prove the following result:

▶ **Theorem 3.** *A factor $\rho \geq 1$ approximation for $L_0$ Fitting Ultrametrics implies a factor $6\rho$ approximation for $L_0$ Fitting Tree Metrics.*

[1] The authors erroneously claim that they get the same approximation for the closest tree metric problem. However, the known reduction may create $\omega(k)$ distinct distances. We believe that the dependence in $k$ is polynomial, which makes the approximation worse, but still non-trivial.

Let $D$ be a distance matrix, $\alpha \in V$ be a distinguished element and $T$ be a tree spanning $V$. In more details, there exists a function mapping elements from $V$ to nodes in $T$. If element $u \in V$ is mapped to node $u' \in T$, we say that $u$ is associated with $u'$. We even say "node $u$" to refer to the node associated with $u$. We note that $T$ may also have auxiliary nodes, without any element from $V$ being mapped to them.

We say $T$ is an $\alpha$-restricted tree if the distance from $\alpha$ to any other element $u$ is the same both in $T$ and in $D$.

Given a tree $T$ we can obtain an $\alpha$-restricted tree $T^{/\alpha}$ by modifying $T$ as in Figure 1. We say that $T^{/\alpha}$ is the $\alpha$-restricted tree of $T$.

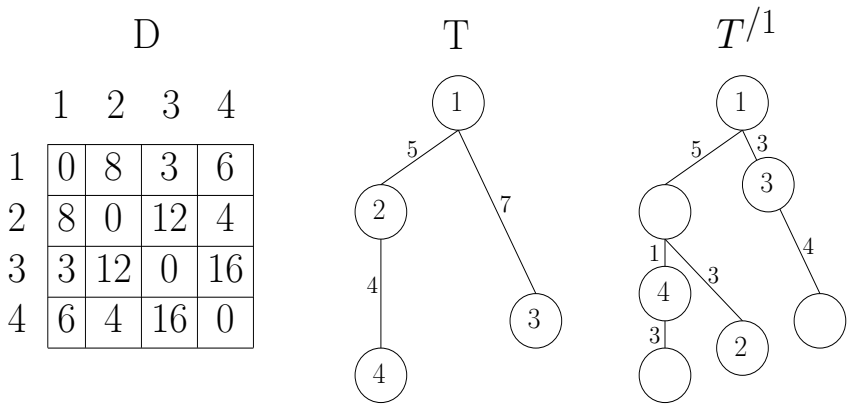



**Figure 1** $T$ is not $\alpha$-restricted, for any $\alpha \in \{1, 2, 3, 4\}$. By modifying $T$ we get $T^{/1}$ which is 1-restricted. Nodes 3 and 4 move towards 1, while 2 moves away from 1 (by creating a new leaf). Notice that some nodes of $T$ may be irrelevant for $T^{/1}$; however we do not need to explicitly delete them.

Intuitively, for any element $u$, if $T(\alpha, u) \neq D(\alpha, u)$, we move $u$ either closer to or further from $\alpha$. More specifically, if $T(\alpha, u) > D(\alpha, u)$, then:

- if there exists a node in the path from $u$ to $\alpha$ at distance $D(\alpha, u)$ from $\alpha$, we associate $u$ with this node,
- else there exists an edge in the path from $u$ to $\alpha$ such that one of its endpoints is at distance less than $D(\alpha, u)$ from $\alpha$, and the other endpoint is at distance larger than $D(\alpha, u)$ from $\alpha$. In this case we subdivide this edge in order to introduce a new node at distance exactly $D(\alpha, u)$ from $\alpha$. Then we associate $u$ with this new node.

Else if $D(\alpha, u) > T(\alpha, u)$ we create a new leaf under the node previously associated with $u$; the length of the edge connecting them is $D(\alpha, u) - T(\alpha, u)$. Then we associate $u$ with the newly created leaf, instead of its parent.

The proof strategy for our main result is the following:

- In order to approximate the optimal tree, it suffices to approximate the optimal $\alpha$-restricted tree, for some $\alpha \in V$. We prove this in Section 2.1.
- In order to approximate the optimal $\alpha$-restricted tree, it suffices to approximate $L_0$ Fitting Constrained Ultrametrics. The proof directly follows from [1]; we include it in the full version for completeness.
- In order to approximate $L_0$ Fitting Constrained Ultrametrics, it suffices to approximate $L_0$ Fitting Ultrametrics. We prove this in Section 2.2.

## 2.1 From tree metrics to restricted tree metrics

The main point where the reduction from [1] breaks for $L_0$ is the connection between optimal tree metrics and optimal $\alpha$-restricted tree metrics. The reason is that an averaging argument used to prove the result uses the convexity of $L_p$-norms, $p \geq 1$. In fact, this argument shows the existence of an element $\alpha$ such that the optimal $\alpha$ restricted tree is very close to the optimal tree. Then one can try all elements $\alpha$ and pick the best of them.

We show that even though $L_0$ is not convex, we can still select $\alpha$ in the same way as in [1]. In particular, we show that for any tree $T$, there exists an element $\alpha$ such that its $\alpha$-restricted tree metric $T^{/\alpha}$ does not increase the cost by more than a factor 3.

▶ **Lemma 4.** *Let $D$ be a distance matrix and $T$ be a tree. Then there exists an element $\alpha$ such that for the $\alpha$-restricted tree $T^{/a}$ of $T$ it holds that $\|D - T^{/a}\|_0 \leq 3\|D - T\|_0$.*

**Proof.** We simply let $\alpha$ be the element that minimizes disagreements, that is

$$\alpha = argmin_{u \in V} \|D(u) - T(u)\|_0$$

where $D(u)$ is the distance matrix $D$ restricted on the pairs containing $u$ (similarly for $T(u)$). We have that $\|D - T\|_0 = \frac{1}{2}\sum_{u \in V} \|D(u) - T(u)\|_0$, as the sum in the right hand side double counts every pair $u, v$ with $D(u, v) \neq T(u, v)$. By definition of $\alpha$, every term in the sum is lower bounded by $\|D(\alpha) - T(\alpha)\|_0$, meaning that

$$\|D - T\|_0 \geq \frac{n}{2}\|D(\alpha) - T(\alpha)\|_0.$$

We say that an element $u$ is good if $D(\alpha, u) = T(\alpha, u)$, and bad otherwise; notice that by definition, the number of bad elements is exactly $\|D(\alpha) - T(\alpha)\|_0$. As any element $u$ can have at most $n - 1$ disagreements in $T^{/\alpha}$ (that is $\|D(u) - T^{/\alpha}(u)\|_0 \leq n - 1$), it follows that the number of pairs $u, v$ with at least one of $u, v$ being bad and $D(u, v) \neq T^{/\alpha}(u, v)$ is at most $\|D(\alpha) - T(\alpha)\|_0 \cdot (n - 1) \leq 2\|D - T\|_0$.

On the other hand, notice that if both $u, v$ are good, then by construction $T^{/\alpha}(u, v) = T(u, v)$. Therefore, if $D(u, v) \neq T^{/\alpha}(u, v)$, it also holds that $D(u, v) \neq T(u, v)$. The number of such pairs is upper bounded by $\|D - T\|_0$. ◀

Letting $T$ be an optimal solution for $L_0$ Fitting Tree Metrics establishes that it suffices to approximate the optimal $\alpha$-restricted tree, only increasing the approximation by a factor 3.

Furthermore, we can directly use the techniques from [1] to show that any approximation for the constrained ultrametrics problem can give the exact same approximation for the optimal $\alpha$-restricted tree. We include this proof in the full version for completeness, as it is itself very brief. However we do not include it here, as it has been extensively used in the literature.

Notice that these results already show that a $\rho$ approximation for $L_0$ Fitting Constrained Ultrametrics translates to a $3\rho$ approximation for $L_0$ Fitting Tree Metrics. This is an extension of the result of [1] for the $L_0$ case.

## 2.2 From constrained ultrametrics to ultrametrics

In this section we show that it is sufficient to approximate $L_0$ Fitting Ultrametrics, which is more natural than $L_0$ Fitting Constrained Ultrametrics. The technique used follows the one used in [5] for $L_p$, $p \in \{1, 2, \ldots\} \cup \{\infty\}$. However, in the case of $L_0$ we can simplify.

The high-level view of the technique is the following: Instead of trying to find a constrained ultrametric close to a distance matrix $D$, we rather squeeze $D$ itself to obey the constraints. Let $S_D$ be the resulting distance matrix. Then we find an (unconstrained) ultrametric $U$

close to this matrix $S_D$; due to the extra structure we imposed on $S_D$, we can only improve $U$ if we again squeeze it to obey the constraints. The resulting ultrametric $S_U$ is a constrained ultrametric, but all we needed to obtain it was a black-box algorithm for general ultrametrics.

We now define the squeezing process more formally. In the $L_0$ Fitting Constrained Ultrametrics problem, for every element $u$ we are given a value $l_u$ which we call $u$'s lower-bound. Furthermore we are given an upper bound $h$.

A constrained ultrametric $U$ shall satisfy that

$$h \geq U(u,v) \geq \max\{l_u, l_v\} \qquad \forall \{u,v\} \in \binom{V}{2}.$$

Given a distance matrix $A$, we define the *squeezed* $A$ as the distance matrix $S_A$ for which $S_A(u,v) = \min\{h, \max\{D(u,v), l_u, l_v\}\}$, for all $\{u,v\} \in \binom{V}{2}$. Intuitively, $S_A$ is obtained by squeezing $A$ to fit the constraints.

We use the well-known characterization of ultrametrics, that $U$ is an ultrametric iff $\forall \{u,v,w\} \in \binom{V}{3} : U(u,v) \leq \max\{U(u,w), U(v,w)\}$.

▶ **Lemma 5.** *A factor $\rho \geq 1$ approximation for $L_0$ Fitting Ultrametrics implies a factor $2\rho$ approximation for $L_0$ Fitting Constrained Ultrametrics.*

**Proof.** Our approach starts with creating $S_D$, the squeezed $D$. Notice that if $U'$ is a constrained ultrametric, then $\|U' - S_D\|_0 \leq \|U' - D\|_0$. This follows because for any $u, v$ it holds that $\max\{l_u, l_v\} \leq U'(u,v) \leq h$, due to $U'$ being a constrained ultrametric. Therefore $D(u,v) = U'(u,v)$ only if $\max\{l_u, l_v\} \leq U'(u,v) \leq h$. But in this case $S_D(u,v) = D(u,v) = U'(u,v)$.

Similarly, suppose we have an ultrametric $U$, and we create the squeezed $S_U$.

With the exact same reasoning, we have

$$\|S_D - S_U\|_0 \leq \|S_D - U\|_0. \tag{1}$$

Our solution to $L_0$ Fitting Constrained Ultrametrics is to first create $S_D$ by squeezing $D$, then obtain ultrametric $U$ by a $\rho$ approximation to $L_0$ Fitting Ultrametrics on $S_D$, and finally obtain $S_U$ by squeezing $U$.

Let $OPT_{D,C}$ be the closest constrained ultrametric to $D$, and $OPT_{S_D}$ be the closest ultrametric to $S_D$. It suffices to show that $\|D - S_U\|_0 \leq 2\rho \|D - OPT_{D,C}\|_0$ and that $S_U$ is indeed an ultrametric.

By definition of $S_D$, and since $OPT_{D,C}$ is constrained, for any two elements $u, v$ it holds that

$$\min\{D(u,v), OPT_{D,C}(u,v)\} \leq S_D(u,v) \leq \max\{D(u,v), OPT_{D,C}(u,v)\}.$$

The proof follows by a straightforward case analysis of the 3 cases $D(u,v) \leq \max\{l_u, l_v\}$, $\max\{l_u, l_v\} < D(u,v) \leq h$, $h < D(u,v)$. Therefore:

- either $D(u,v) = OPT_{D,C}(u,v)$, in which case

$$|D(u,v) - OPT_{D,C}(u,v)|^0 = 0 = |D(u,v) - S_D(u,v)|^0 + |S_D(u,v) - OPT_{D,C}(u,v)|^0$$

- or $D(u,v) \neq OPT_{D,C}(u,v)$, in which case

$$|D(u,v) - OPT_{D,C}(u,v)|^0 = 1, |D(u,v) - S_D(u,v)|^0 + |S_D(u,v) - OPT_{D,C}(u,v)|^0 \leq 2.$$

We conclude that

$$|D(u,v) - S_D(u,v)|^0 + |S_D(u,v) - OPT_{D,C}(u,v)|^0 \leq 2|D(u,v) - OPT_{D,C}(u,v)|^0. \quad (2)$$

We now have

$$\begin{aligned}
\|D - S_U\|_0 &\leq \|D - S_D\|_0 + \|S_D - S_U\|_0 && \text{(triangle inequality)} \\
&\leq \|D - S_D\|_0 + \|S_D - U\|_0 && (1) \\
&\leq \|D - S_D\|_0 + \rho\|S_D - OPT_{D,C}\|_0. && \text{(definition of } U\text{)}
\end{aligned}$$

As $\rho \geq 1$, the latter is upper bounded by

$$\begin{aligned}
&\rho \sum_{\{u,v\} \in \binom{V}{2}} (|D(u,v) - S_D(u,v)|^0 + |S_D(u,v) - OPT_{D,C}(u,v)|^0) \\
&\leq 2\rho \sum_{\{u,v\} \in \binom{V}{2}} (|D(u,v) - OPT_{D,C}(u,v)|^0) && (2) \\
&= 2\rho\|D - OPT_{D,C}\|_0.
\end{aligned}$$

Finally, we need to prove that $S_U$ inherits that it is an ultrametric. This is clear if we proceed in rounds; each round we construct a new ultrametric, and the last one will coincide with $S_U$.

More formally, let $U_0 = U$. In the first $|V|$ rounds, we take out a different $u' \in V$ at a time, and let

$$U_r(u', v) = \max\{U_{r-1}(u', v), l_{u'}\} \qquad \forall v \neq u'.$$

Suppose $r > 0$ is the first round where $U_r$ is not an ultrametric. Then there exists a triplet $\{u, v, w\}$ such that $U_r(u, v) > \max\{U_r(u, w), U_r(v, w)\}$. As we only increase distances, this may only happen if $U_r(u, v) > U_{r-1}(u, v)$. But this means that at round $r$ we picked either $u$ or $v$ (w.l.o.g. assume it was $u$) and set $U_r(u, v) = l_u$. However, this would also give $U_r(u, w) \geq l_u = U_r(u, v)$, contradicting $U_r(u, v) > \max\{U_r(u, w), U_r(v, w)\}$.

Finally, for $S_U$ we simply have

$$S_U(u, v) = \min\{h, U_{|V|}(u, v)\}.$$

Suppose there exists a triplet $\{u, v, w\}$ that now violates the ultrametric property, then it holds that

$$S_U(u, v) > \max\{S_U(u, w), S_U(v, w)\}.$$

As we did not increase any distance of $U_{|V|}$, this means that $S_U(u, w) < U_{|V|}(u, w)$ or $S_U(v, w) < U_{|V|}(v, w)$; but distances can only reduce to $h$ which is an upper bound on $S_U(u, v)$ by construction. ◀

To prove our main theorem we use the following result from [1]. We only provide its proof in the full version, for completeness.

▶ **Lemma 6** (Implicit in the proof of Lemma 3.5 of [1]). *Let $D$ be a distance matrix, $\alpha$ be an element, and $T$ be an $\alpha$-restricted tree metric minimizing $\|T - D\|_0$. Assuming a $\gamma$-approximation to $L_0$ Fitting Constrained Ultrametrics, we can find an $\alpha$-restricted tree $T'$ such that $\|T' - D\|_0 \leq \gamma\|T - D\|_0$.*

We are now ready to prove our main theorem.

▶ **Theorem 3.** *A factor $\rho \geq 1$ approximation for $L_0$ Fitting Ultrametrics implies a factor $6\rho$ approximation for $L_0$ Fitting Tree Metrics.*

**Proof.** We iterate over all $u \in V$ nodes. In every iteration we use Lemma 6 along with a $2\rho$ approximation for $L_0$ Fitting Constrained Ultrametrics (obtained by Lemma 5) to obtain a tree $T_u$ with $\|T_u - D\|_0 \leq 2\rho\|T'_u - D\|_0$, where $T'_u$ is the optimal $u$-restricted tree metric. Out of all the trees $T_u$ that we obtain, we output $T$, the one that minimizes $\|T_u - D\|_0$.

Let $T_{OPT}$ be an optimal tree metric. By Lemma 4 there exists an element $\alpha$ such that for the $\alpha$-restricted tree $T_{OPT}^{/\alpha}$ of $T_{OPT}$ it holds that $\|T_{OPT}^{/\alpha} - D\|_0 \leq 3\|T_{OPT} - D\|_0$. Therefore there exists an element $\alpha$ for which we have $\|T'_\alpha - D\|_0 \leq 3\|T_{OPT} - D\|_0$.

It now holds that $\|T - D\|_0 \leq \|T_\alpha - D\|_0 \leq 2\rho\|T'_\alpha - D\|_0 \leq 6\rho\|T_{OPT} - D\|_0$. ◀

▶ **Corollary 7.** *There exists a polynomial time $O(1)$ approximation for $L_0$ Fitting Tree Metrics.*

**Proof.** Follows immediately, by using the polynomial time $O(1)$ approximation for $L_0$ Fitting Ultrametrics from [6]. ◀

## 3 APX-Hardness

In this section we show that $L_0$ Fitting Tree Metrics is APX-Hard. Assuming, for the sake of contradiction, that it is not the case, we show how to approximate Correlation Clustering (an APX-Hard problem [4]) within any constant factor.

This is a standard reduction used for $L_p$ Fitting Tree Metrics, $p \geq 1$. It is however further simplified for $L_0$. That is because once we decide to move a node, our cost does not depend on the distance we moved it.

In Correlation Clustering, we are given an unweighted undirected graph $G$, and the goal is to output a partition of the vertices[2] (clustering) such that we minimize the number of pairs of vertices connected by an edge in $G$ that are in different parts of the partition (clusters) plus the number of pairs of vertices not connected by an edge in $G$ that are in the same part of the partition.

The idea behind the reduction is the following: for every pair of vertices connected by an edge in $G = (V, E)$, we set their distance to 0, and for every pair of vertices not connected by an edge, we set their distance to something larger (2 in our case). Then we solve $L_0$ Fitting Tree Metrics. If the output tree has a good structure (every pair of nodes associated with elements of $V$ has equal distance), it would directly correspond to a clustering. Namely, each node associated with elements of $V$ corresponds to a cluster that contains all elements associated with it (may be more than one). Even though we can guarantee that an optimal solution has this structure, our approximation may not.

To fix this, we introduce many more elements at distance 0 from each other, and at distance 1 from every element in $V$. This has the effect of maintaining the structure of an optimal solution, while incuring a big error in every solution that does not have the desired structure.

We now formally prove our result.

[2] To avoid confusion, we use the term vertices when we refer to Correlation Clustering, and nodes when we refer to a tree.

▶ **Theorem 8.** *$L_0$ Fitting Tree Metrics is APX-Hard.*

**Proof.** Let $G = (V, E)$ be the input to Correlation Clustering, and $\epsilon > 0$ be a sufficiently small constant. For the sake of contradiction we assume that $L_0$ Fitting Tree Metrics can be approximated within an $1 + \epsilon$ factor. Then we show that using this approximation, we can approximate Correlation Clustering within the same factor.

Let $V'$ be a set, disjoint from $V$, of size $|V'| = 2\binom{|V|}{2}$. For any two elements $u', v' \in V'$ we have $D(u', v') = 0$. For $u, v \in V$ we have that $D(u, v) = 0$ if $\{u, v\} \in E$, and 2 otherwise. Finally, for $u \in V, u' \in V'$ we have $D(u, u') = 1$. Let $T$ be the tree output by $L_0$ Fitting Tree Metrics on $D$.

An upper bound for the optimal value is $\binom{|V|}{2}$. To see this, create the tree $T'$ consisting of two nodes $u_{T'}, v_{T'}$ at distance 1. All elements of $V'$ are associated with $u_{T'}$, and all elements of $V$ are associated with $v_{T'}$. As the only disagreements between $T'$ and $D$ are pairs of elements of $V$, the upper bound follows.

Furthermore, any solution that does not have all elements of $V'$ associated with the same node in $T$ has cost at least $|V'| - 1$. Therefore, for sufficiently small $\epsilon$, $T$ must have all elements of $V'$ associated with the same node $v'$. Similarly, all elements of $V$ must be associated with nodes at distance 1 from $v'$. In particular, this corresponds to an ultrametric, where the root node is $v'$, and all leaves are at depth 1. In what follows we consider this tree rooted at $v'$.

Finally, if any non-root node of $T$ is at distance less than 1 from $v'$, we remove it by connecting all its children with its parent node. Notice that this does not increase the number of disagreements, because the distance between elements of $V$ is either 0 or 2. After we can no longer remove any node, we are left with a tree $T$ with root $v'$, and children $v_1, \ldots, v_\ell$ at distance 1 from $v'$ (thus distance 2 from each other). Each $v_i$ is associated with some elements from $V$. Our solution to Correlation Clustering is the partition of $V$ induced by $v_1, \ldots, v_\ell$.

By construction of $D$, the cost of this Correlation Clustering solution is exactly equal to $\|T - D\|_0$. Furthermore, if $C_1, \ldots, C_{\ell'}$ is the optimal solution to Correlation Clustering, we can create a tree with a root $v'$ and children $v_1, \ldots, v_{\ell'}$ such that all elements of $V'$ are associated with $v'$ and all elements of $C_i$ are associated with $v_i$. This means that the optimal Correlation Clustering cost is an upper bound to the optimal $L_0$ Fitting Tree Metrics.

We conclude that we found an $1+\epsilon$ approximation for Correlation Clustering, contradicting its APX-Hardness. ◀

We note that the proof assumes some distances to be 0. If we want distances to be strictly positive, we can select a sufficiently small constant $\delta$ instead of 0. Then, we replace nodes that are associated with multiple elements with stars whose leaves all have distance $\delta$ to each other.

## References


1 Richa Agarwala, Vineet Bafna, Martin Farach, Mike Paterson, and Mikkel Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). *SIAM J. Comput.*, 28(3):1073–1085, 1999. Announced at SODA 1996.

2 Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. *SIAM J. Comput.*, 40(5):1275–1291, 2011. Announced at FOCS 2005.

3 Gunnar E. Carlsson and Facundo Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *J. Mach. Learn. Res.*, 11:1425–1470, 2010.

4 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. Announced at FOCS 2003.

5 Vincent Cohen-Addad, Debarati Das, Evangelos Kipouridis, Nikos Parotsidis, and Mikkel Thorup. Fitting distances by tree metrics minimizing the total error within a constant factor. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 468–479. IEEE, 2021.

6 Vincent Cohen-Addad, Chenglin Fan, Euiwoong Lee, and Arnaud de Mesmay. Fitting metrics and ultrametrics with minimum disagreements. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 301–311. IEEE, 2022.

7 Martin Farach, Sampath Kannan, and Tandy J. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13(1/2):155–179, 1995. Announced at STOC 1993.

8 Boulos Harb, Sampath Kannan, and Andrew McGregor. Approximating the best-fit tree under $l_p$ norms. In *APPROX-RANDOM*, pages 123–133, 2005.

9 M.S. Waterman, T.F. Smith, M. Singh, and W.A. Beyer. Additive evolutionary trees. *Journal of Theoretical Biology*, 64(2):199–213, 1977.

# Coloring Tournaments with Few Colors: Algorithms and Complexity

**Felix Klingelhoefer**
Laboratoire G-SCOP (Univ. Grenoble Alpes), Grenoble, France

**Alantha Newman**
Laboratoire G-SCOP (CNRS, Univ. Grenoble Alpes), Grenoble, France

## Abstract

A *k-coloring* of a tournament is a partition of its vertices into $k$ acyclic sets. Deciding if a tournament is 2-colorable is NP-hard. A natural problem, akin to that of coloring a 3-colorable graph with few colors, is to color a 2-colorable tournament with few colors. This problem does not seem to have been addressed before, although it is a special case of coloring a 2-colorable 3-uniform hypergraph with few colors, which is a well-studied problem with super-constant lower bounds.

We present an efficient decomposition lemma for tournaments and show that it can be used to design polynomial-time algorithms to color various classes of tournaments with few colors, including an algorithm to color a 2-colorable tournament with ten colors. For the classes of tournaments considered, we complement our upper bounds with strengthened lower bounds, painting a comprehensive picture of the algorithmic and complexity aspects of coloring tournaments.

**2012 ACM Subject Classification** Mathematics of computing → Approximation algorithms

**Keywords and phrases** Tournaments, Graph Coloring, Algorithms, Complexity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.71

**Related Version** *Full Version*: https://arxiv.org/abs/2305.02922 [28]

**Funding** Supported in part by ANR project DAGDigDec (ANR-21-CE48-0012).

**Acknowledgements** We thank Louis Esperet for useful discussions and for his encouragement.

## 1 Introduction

A tournament $T = (V, A)$ is a complete, oriented graph: For each pair of vertices $i, j \in V$, there is either an arc from $i$ to $j$ or an arc from $j$ to $i$ (but not both). A subset of vertices $S \subseteq V$ induces the *subtournament* $T[S]$. If this subtournament contains no directed cycles, then it is said to be *acyclic*. The problem of *coloring a tournament* is that of partitioning the vertices into the minimum number of acyclic sets, sometimes referred to as the *dichromatic number* [32]. Since a tournament contains a directed cycle if and only if it contains a directed triangle, the problem of coloring a tournament is equivalent to partitioning the vertices into the minimum number of sets so that each set does not contain a directed triangle.

Coloring tournaments can be compared to the problem of coloring undirected graphs. For the latter, deciding if a graph is 2-colorable (i.e., bipartite) is easy, but it is NP-hard to decide if a graph is 3-colorable. A widely-studied promise problem is that we are given a graph promised to be 3-colorable and the goal is to color it (in polynomial time) with few colors [34, 5, 23, 24]. For tournaments, it is easy to decide whether or not a tournament is 1-colorable (i.e., transitive), since this is exactly when the tournament is acyclic. However, deciding if a tournament is 2-colorable is already NP-hard [8].

This suggests the following promise problem: Given a tournament promised to be 2-colorable, what is the fewest number of colors with which it can be colored in polynomial time? This question is the starting point for this paper and naturally leads to related problems of determining upper and lower bounds for coloring various classes of tournaments. For


**Table 1** Best known lower and upper bounds for various graph coloring problems. All inapproximability results are under the assumption P $\neq$ NP except those denoted by $^*$, which are under the $d$-To-1 Conjecture [26]. The lower bound should be read as, "It is hard to color a 3-colorable graph with 5 colors." The upper bound as, "A 3-colorable graph can be (efficiently) colored with $\tilde{O}(n^{0.19996})$ colors."

| Graph Type | Lower Bound | Upper Bound |
|---|---|---|
| 3-Colorable graphs | 5 [6], $O(1)^*$ [18] | $\tilde{O}(n^{0.19996})$[24] |
| $k$-Colorable graphs, $k \geq 3$ | $2k-1$ [6], $O(1)^*$[18] | $O(n^{1-\frac{3}{k+1}})$ [23] |
| General graphs | $n^{1-\epsilon}$ [22, 35] | $O(n(\log\log n)^2(\log n)^{-3})$ [19] |
| 3-Uniform 2-colorable hypergraphs | $O(1)$ [11] | $\tilde{O}(n^{\frac{1}{5}})$ [29] |

comparison, the complexity landscape of graph coloring is well studied and we have a general understanding of what it looks like. (See Table 1.) In contrast, the problem of coloring tournaments has been studied very little from the algorithmic or complexity perspective. This paper is an effort to address this disparity.

## 1.1 Previous Work

The problem of coloring a 2-colorable tournament with few colors is a special case of coloring a 2-colorable 3-uniform hypergraph with few colors. Deciding if a 3-uniform hypergraph is 2-colorable is NP-hard [31] and more recently it was proved to be NP-hard to color with any constant number of colors [11]. On the positive side, a 2-colorable 3-uniform hypergraph can be colored in polynomial time with $\tilde{O}(n^{1/5})$ colors [1, 7, 29], a result which uses tools from and is analogous to that of [23] for 3-colorable graphs. Thus, $\tilde{O}(n^{1/5})$ is the best-known upper bound on the number of colors needed to efficiently color a 2-colorable tournament. Deciding if a tournament is 2-colorable is NP-hard [8] and furthermore, deciding if a tournament is $k$-colorable for any $k \geq 2$ is NP-hard [15]. It is consistent with these results that we can, say, efficiently color a 2-colorable tournament with three colors.

From a structural graph theory perspective, the problem of coloring tournaments has been widely studied due to its connection to the famous Erdős-Hajnal Conjecture [12, 9], which has an equivalent formulation in terms of tournaments [2]. The latter posits that for any tournament $H$, there is a constant $\epsilon_H$ (where $0 < \epsilon_H \leq 1$) such that any $H$-free tournament on $n$ vertices has a transitive subtournament of size at least $O(n^{\epsilon_H})$. [4] exactly characterize the tournaments for which $\epsilon_H = 1$, which they call *heroes*. Forbidding a hero in a tournament $T$ actually results in $T$ being colorable with a constant number of colors [4], which yields a transitive induced subtournament of linear size. These results are existential and do not provide an efficient algorithm to color an $H$-free tournament with a constant number of colors, when $H$ is some fixed hero.

## 1.2 Our Results

We consider some basic algorithmic and computational complexity questions on the subject of coloring tournaments. Our main algorithmic tool, presented in Section 2, is a decomposition lemma which can be used to obtain efficient algorithms for coloring tournaments in various cases when certain conditions are met. On a high level, it bears some resemblance to decompositions previously used to prove bounded dichromatic number in tournaments and in dense digraphs with forbidden subgraphs [4, 20]. To apply our decomposition lemma to 2-colorable tournaments, we use an observation used by [1, 7, 29] which states that

**Table 2** Best known polynomial time inapproximability results and approximation algorithms for various tournament coloring problems. Previous results are indicated with a citation. All the results without a citation are established in this paper. Lower bounds are under the assumption $\mathsf{P} \neq \mathsf{NP}$ except those marked with a $^*$, which hold under the $d$-To-1 Conjecture [26]. The function $g(k)$ denotes the number of colors needed to efficiently color a $k$-colorable graph, while $f(k)$ is the number of colors needed to efficiently color a $k$-colorable tournament. The entry indicated by $^\dagger$ is a hardness of approximation result.

| Tournament Type | Lower Bound | Upper Bound |
|---|---|---|
| 2-Colorable tournaments | 2[8], 3 | 10 |
| 3-Colorable tournaments | 5, $O(1)$ $^*$ | $\tilde{O}(n^{0.19996})$ |
| $k$-Colorable tournaments, $k \geq 2$ | $2k-1$, $O(1)$ $^*$ | $5 \cdot f(k-1) \cdot g(k)$ |
| 2-Colorable light tournaments | in P? | 5 |
| Light tournaments | in P? | 9 |
| General tournaments | $n^{\frac{1}{2}-\epsilon}$ $^\dagger$ | $n/\log n$[13] |

there is an efficient algorithm to partition a 2-colorable tournament into two tournaments that are each light. A *light tournament* is one in which for each arc $uv$, the set of vertices $N(uv) = \{w \mid uvw$ forms a directed triangle$\}$ is transitive. (Let $C_3$ denote a directed triangle. A light tournament is $H$-free where $H$ is the hero $(C_3, 1, 1)$.)

In fact, due to this observation and the fact that [4] showed that light tournaments have constant dichromatic number, it cannot be NP-hard (unless NP= co-NP) to color a 2-colorable tournament with $O(1)$ colors. (This does not however immediately imply that there is an efficient algorithm, since there are many search problems that are believed to be intractable even though their decision variant is easy, e.g., those in the class TFNP.) Although [4] did not provide an efficient algorithm to color a light tournament with a constant number of colors, a careful modification of their techniques indeed results in a polynomial-time algorithm using around 35 colors to color a light tournament.

Like some other lemmas which show that the dichromatic number of a tournament is bounded (i.e., constant) if the out-neighborhoods of vertices have bounded dichromatic number [21], our decomposition lemma also has a local-to-global flavor: If the sets $N(uv)$ can be efficiently colored with few colors for all arcs $uv$ and if there are two vertices $s$ and $t$ such that the out-neighborhood of $s$ and the in-neighborhood of $t$ can be efficiently colored with few colors, then our decomposition lemma yields an efficient algorithm to color the whole tournament with few colors.

We give applications of our algorithmic decomposition lemma in Section 3. Specifically, we show that 2-colorable tournaments can be efficiently colored with ten colors and that light tournaments can be efficiently colored with nine colors. We then use our toolbox to study 3-colorable tournaments. Here we show that the problem of coloring a 3-colorable tournament has a constant-factor reduction to the problem of coloring 3-colorable graphs.

Next, we strengthen the lower bounds by showing in Section 4 that it is NP-hard to color a 2-colorable tournament with three colors. We then give a reduction from coloring graphs to coloring tournaments, which implies, for example, that it is hard to color 3-colorable tournaments with $O(1)$ colors under the $d$-To-1 Conjecture of Khot [26]. Finally, we show that it is NP-hard to approximate the number of colors required for a general tournament to within a factor of $O(n^{1/2-\epsilon})$ for any $\epsilon > 0$. Our results are summarized in Table 2.

## 1.3 Notation and Preliminaries

Let $T = (V, A)$ be a tournament with vertex set $V$ and arc set $A$. Sometimes, we use $V(T)$ to denote its vertex set and $A(T)$ to denote its arc set. For $S \subset V$, we use $T[S]$ to denote the subtournament induced on vertex set $S$, although we sometimes abuse notation and refer to the subtournament itself as $S$. We define $uv \in A$ to be an arc directed from $u$ to $v$. We define $N^+(v)$ to be all $w \in V$ such that arc $vw \in A$ and $N^-(v)$ to be all $w \in V$ such that arc $wv \in A$. We let $N^+[v] = N^+(v) \cup \{v\}$ and $N^-[v] = N^-(v) \cup \{v\}$. For $S \subset V$, we define $N^+(S) = \bigcup_{v \in S} N^+(v)$, and we define $N^-(S), N^+[S], N^-[S]$ analogously. We use $N^{\pm}(S)$ to denote vertices in $V \setminus S$ that have at least one in-neighbor and at least one out-neighbor in $S$. Sometimes we refer to $N^{\pm}(S)$ of a set as its *mixed neighborhood*.

For $S, U \subset V$ such that $S \cap U = \emptyset$, we use $S \Rightarrow U$ to indicate that all arcs between $S$ and $U$ are directed from $S$ to $U$. Let $C_3$ denote a directed triangle; usually, we refer to this simply as a triangle. Define $N(uv) \subset V$ to contain all vertices $w$ such that $uvw$ forms the directed triangle consisting of arcs $uv, vw$ and $wu$. In other words, $N(uv) = N^-(u) \cap N^+(v)$. For three tournaments $T_1, T_2$ and $T_3$, we use $\Delta(T_1, T_2, T_3)$ to denote the tournament resulting from adding all arcs from $T_1$ to $T_2$, all arcs from $T_2$ to $T_3$ and all arcs from $T_3$ to $T_1$.

A tournament $T = (V, A)$ is *$k$-colorable* if there is a partition of $V$ into $k$ vertex-disjoint sets, $V_1, V_2, \ldots, V_k$, such that $T[V_i]$ is transitive for all $i \in \{1, \ldots, k\}$. We use $\vec{\chi}(T)$ to denote the *dichromatic number* of $T$ (i.e., the minimum number of transitive subtournaments into which $V(T)$ can be partitioned). Computing the value $\vec{\chi}(T)$ is in general NP-hard [8]. We therefore use $\vec{\chi}_{\mathcal{C}}(T)$ to denote the number of colors by which $T$ can be efficiently colored. Our goal is to find upper and lower bounds on $\vec{\chi}_{\mathcal{C}}(T)$.

We remark that we will always assume that a tournament $T$ which we want to color is strongly connected; if this were not the case, we can color each strongly connected component separately. Therefore, each vertex has an out-neighborhood containing at least one vertex.

## 2 Efficient Tournament Decomposition for Coloring

We present a decomposition for a tournament that can be computed in polynomial time and yields an efficient method to color a tournament tournaments with few colors in certain cases.

▶ **Definition 1.** *We define a* $c$-vertex chain $(v_i)_{0 \le i \le k}$ *of a tournament* $T$ *the following way: Let* $v_0$ *and* $v_k$ *be a pair of vertices such that* $\vec{\chi}_{\mathcal{C}}(N^+(v_0) \cup N^-(v_k)) \le c$*, and let* $(v_i)_{0 \le i \le k}$ *be the vertices in the shortest directed path from* $v_0$ *to* $v_k$*.*

Additionally, we define an *arc chain* $(e_i)_{1 \le i \le k}$ corresponding to a vertex chain, where $e_i$ is the arc from $v_{i-1}$ to $v_i$. The main idea behind this decomposition is to build zones that can be efficiently colored, and such that all arcs between zones at distance more than four (i.e., *long* arcs) go backwards.

▶ **Definition 2.** *Given a* $c$*-vertex chain, a* path decomposition *of a tournament* $T$ *is defined as:*

- $D_0 = N^+(v_0)$.
- *For* $1 \le i \le k$, $D_i = N(e_i) \setminus (\cup_{0 \le j \le i-1} D_j)$.
- $D_{k+1} = N^-(v_k) \setminus (\cup_{0 \le j \le k} D_j)$.

First we prove that this is indeed a decomposition of $T$.

▶ **Lemma 3.** *Let* $T = (V, A)$ *be a tournament and let* $(D_0, \ldots, D_{k+1})$ *be a path decomposition of* $T$*. Then* $V = \cup_{0 \le i \le k+1} D_i$.

**Proof.** We will prove this lemma by contradiction: Suppose there is a vertex $w \in V$ that does not belong to any $D_i$. Assume that $w$ does not belong to the vertex chain. Since $w$ is neither in $D_0$ nor in $D_{k+1}$, then $w \in N^-(v_0)$ and $w \in N^+(v_k)$. Take the smallest integer $i$ such that $w \in N^+(v_i)$. There must be one since $w \in N^+(v_k)$. Notice that $i \geq 1$ since $w \notin N^+(v_0)$, so $e_i$ belongs to the arc chain and $w \in N(e_i)$. Therefore, $w \in D_i$, which is a contradiction.

Now consider the case in which $w$ is in the vertex chain. An arc with both endpoints in the vertex chain that is not in the arc chain is backwards. Thus, $v_i \in N(e_{i+2})$ for all $0 \leq i \leq k-2$. Notice that $v_{k-1}$ can belong to $D_{k+1}$ (if it does not belong to $D_j$ for some $j < k+1$). Finally, $v_k \in N(e_{k-1})$. ◀

We remark that, for the sake of simplicity and to more easily visualize the decomposition, it might be easier to not include the vertices in the vertex chain in the path decomposition. In this case, these vertices can be colored with two extra colors. Since all arcs not in the arc chain with both endpoints in the vertex chain go backwards (with respect to the arc chain; otherwise there would be an even shorter path), we can use two colors so that all forwards arcs (those in the arc chain) are bicolored.

▶ **Lemma 4.** *Let $0 \leq i, j \leq k+1$ and let $j \geq i+5$. For $u \in D_i$ and $w \in D_j$, we have $u \in N^+(w)$.*

**Proof.** We will prove this by contradiction. Suppose $j \geq i+5$ and $u \in N^-(w)$. Then there is a path of three arcs from $v_i$ to $v_{j-1}$, namely $(v_i, u, w, v_{j-1})$. (By definition of the decomposition, $u \in D_i$ implies $u \in N^+(v_i)$ and $w \in D_j$ implies $w \in N^-(v_{j-1})$.) This is not possible since by the definition of the vertex chain as the shortest path, there can be no path between $v_i$ and $v_{j-1}$ with fewer than four arcs (since $(j-1)-i \geq (i+5-1)-i = 4$). ◀

▶ **Lemma 5.** *If $T$ has a $c$-vertex chain that can be found in polynomial time and if $\vec{\chi}_{\mathcal{C}}(N(e)) \leq c$ for each arc $e$ in the corresponding arc chain, then $\vec{\chi}_{\mathcal{C}}(T) \leq 5c$.*

**Proof.** Given a $c$-vertex chain, we construct a path decomposition. We make five palettes of $c$ colors each with labels from 0 to 4. We color each $D_i$ using the color palette with label $i$ mod 5. Let us show that we can do this in polynomial time. First, note that the set of colors used is of size $c$ for every $D_i$. Then, let us consider $D_0$: $N^+(v_0)$ can be colored efficiently with $c$ colors by definition of a vertex chain. Similarly, $D_{k+1}$ is a subset of $N^-(v_k)$ and can thus also be efficiently colored with $c$ colors. Finally, for every $1 \leq i \leq k$, $D_i$ is a subset of $N(e_i)$, which can be colored efficiently with $c$ colors by the condition of the lemma.

Our goal is now to prove that this is a proper coloring of $T$. We will do this by showing that all forward arcs between different $D_i$ are bicolored. By Lemma 4, there are no forwards arcs between $D_i$ and $D_j$ when $j \geq i+5$. Furthermore, by the definition of the coloring, no vertex in $D_i$ and $D_j$ can share a color for $i+1 \leq j \leq i+4$. Thus all forward arcs from $D_i$ to $D_j$ will be bicolored. Since every $D_i$ is properly colored, and all forward arcs between different $D_i$ are bicolored, $T$ is properly colored. ◀

The next lemma has essentially the same proof as Lemma 5.

▶ **Lemma 6.** *If $T$ has a $c$-vertex chain that can be found in polynomial time and if $\vec{\chi}_{\mathcal{C}}(N(e)) \leq d$ for each arc $e$ in the arc chain and if $c > d$, then $\vec{\chi}_{\mathcal{C}}(T) \leq c + 4d$.*

**Proof.** We find the path decomposition using the $c$-vertex chain. We can color the set $S = D_0 \cup D_{k+1}$ with $c$ colors and the remaining sets $D_i$ for $1 \leq i \leq k$ with $d$ colors each. For the last $c-d$ of the colors used for $S$, we can remove these vertices from $S$ since these

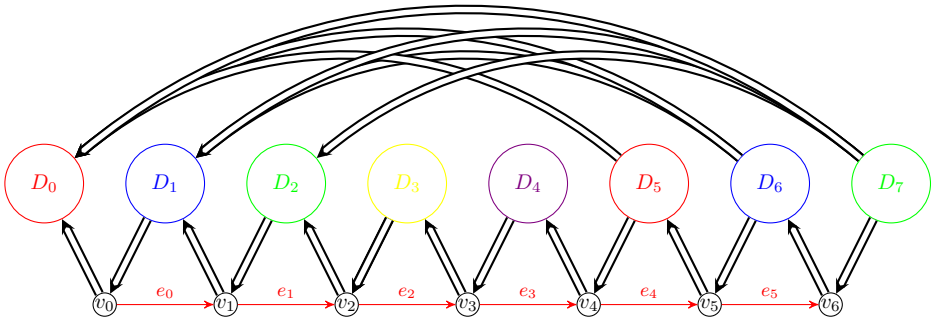



**Figure 1** A path decomposition of $T$. The red arcs $(e_i)$ form a shortest path from $v_0$ to $v_k$, thus all the arcs not depicted between the $v_i$'s go backward. All the vertices in a given $D_i$ are colored from the color palette indicated by the color of the $D_i$. Notice that because there are no long forward arcs between the $D_i$'s, all arcs between $D_i$'s that share a color palette are backwards.

colors will not be used again and call the remaining vertices in $S$ (colored with the first $d$ colors) $S'$. For the remaining vertices in $S$, we decompose them into $D_0 := D_0 \cap S'$ and $D_{k+1} := D_{k+1} \cap S'$ Now we have sets $D_0, D_1, \ldots, D_{k+1}$ each colored with $d$ colors. We color these sets using five color palettes of $d$ colors each and use the palette $i \bmod 5$ for set $D_i$. By Lemma 4, this does not create any monochromatic forward arcs. Thus, the total number of colors used is $(c-d)+5d = c+4d$. ◀

## 3 Algorithms for Coloring Tournaments

We consider various special cases of tournaments and show how to use our tools to color them with few colors.

### 3.1 2-Colorable Tournaments

A tournament $T = (V, A)$ is *2-colorable* if $\vec{\chi}(T) = 2$, and a 2-coloring of tournament $T$ is a partition of $V$ into two vertex sets, $V_1$ and $V_2$, such that $T[V_1]$ and $T[V_2]$ are each transitive. In this section, our goal is to prove Theorem 7.

▶ **Theorem 7.** *Let $T$ be a 2-colorable tournament. Then $\vec{\chi}_C(T) \leq 10$.*

We say an arc $uv$ in $A$ is *heavy* if there exist three vertices $a, b, c \in N(uv)$ which form a triangle $abc$. If a tournament contains no heavy arcs, then it is *light*. We will use the following observation.

▶ **Observation 8.** *Let $T$ be a 2-colorable tournament. Then $T$ can be partitioned into two light subtournaments $T_1$ and $T_2$ such that $\vec{\chi}_C(T) \leq \vec{\chi}_C(T_1) + \vec{\chi}_C(T_2)$.*

This observation appears in [1, 7, 29] where it is stated more generally for 2-colorable 3-uniform hypergraphs. We include a proof here for completeness.

▶ **Lemma 9.** *In a 2-coloring of a tournament $T$, each heavy arc must be 2-colored.*

**Proof.** If $u$ and $v$ are both, say, blue, then each vertex in $N(uv)$ would be red, forcing a triangle in $N(uv)$ to be all red (i.e., monochromatic), which is not possible in a 2-coloring. ◀

▶ **Corollary 10.** *In a 2-colorable tournament, the heavy arcs form a bipartite graph.*

Now we can prove Observation 8.

**Proof of Observation 8.** All heavy arcs can be easily detected. By Corollary 10, the set of heavy arcs forms a bipartite graph. The vertex set of this bipartite graph can be colored with two colors (red and blue), such that the tournament induced by each color does not contain a heavy arc. Then we partition the vertices into two sets one containing all the blue vertices and the other containing all the red vertices. The uncolored vertices can go in either set. Since neither of these sets contains any heavy arcs, we can partition the vertices of a 2-colorable tournament into two light subtournaments. ◀

Theorem 7 will follow from Observation 8 and the following theorem.

▶ **Theorem 11.** *Let $T$ be a 2-colorable light tournament. Then $\vec{\chi}_C(T) \leq 5$.*

Our goal it to use Lemma 5 to prove Theorem 11. In other words, we want to show that a 2-colorable light tournament has a 1-vertex chain. We first prove a useful claim.

▶ **Lemma 12.** *Let $T$ be a $k$-colorable tournament. Then there exist vertices $u$ and $w$ such that $N^+(u) \cup N^-(w)$ is $(k-1)$-colorable.*

**Proof.** Since $T = (V, A)$ is $k$-colorable, there exist $k$ transitive sets $X_1, \ldots, X_k$ such that $V = \cup_{i=1}^{k} X_i$. Then take $u$ to be the vertex in $X_1$ that has only incoming arcs from other vertices in $X_1$ (i.e., the sink vertex for $X_1$). Similarly, take $w$ to be the vertex in $X_1$ that has only outgoing arcs to other vertices in $X_1$ (i.e., the source vertex for $X_1$). The out-neighborhood of $u$ and the in-neighborhood of $w$ are both subsets of $V \setminus X_1$, and thus so is their union, which is therefore $(k-1)$-colorable. ◀

Now we are ready to prove that we can find a 1-vertex chain.

▶ **Lemma 13.** *Let $T$ be a 2-colorable, light tournament. Then $T$ contains a 1-vertex chain that can be found in polynomial time.*

**Proof.** By Lemma 12, there exist $u$ and $w$ such that $N^+(u) \cup N^-(w)$ is transitive. To find them, we can test the transitivity of $N^+(u) \cup N^-(w)$ for every pair of vertices in $T$. Then we simply need to find a shortest path from $u$ to $w$, which can be done in polynomial time. Let $k$ denote the length of the path, and define $v_0 = u$, $v_k = w$, and $(v_i)_{1 \leq i \leq k-1}$ the rest of the vertices in the path. ◀

The proof of Theorem 11 follows from Lemma 13, Lemma 5 and the fact that $\vec{\chi}_C(N(e)) \leq 1$ for every arc $e$ in a light tournament.

## Certificates of Non-2-Colorability

In Section 3.1, we presented an algorithm to color a 2-colorable tournament with ten colors. Suppose we run this algorithm on an arbitrary tournament $T$ (e.g., one that is *not* 2-colorable). Then our algorithm will either color $T$ with ten colors or it will produce at least one certificate that $T$ is not 2-colorable. A certificate will have the following form: either a) there is an odd cycle of heavy arcs in $T$, or b) for every ordered pair of vertices $(u, v)$, the subtournament $T[N^+(u) \cup N^-(v)]$ is not transitive. In particular, an 11-chromatic tournament must contain such a certificate.

## 3.2 3-Colorable Tournaments

Coloring 3-colorable tournaments turns out to be closely related to coloring 3-colorable graphs. This seems surprising since the techniques for 3-colorable graphs were applied to coloring 2-colorable 3-uniform hypergraphs, which are a generalization of 2-colorable tournaments.

We will first show that we can adapt ideas of [34] and [5] to the problem of coloring 3-colorable tournaments by using our algorithm for coloring 2-colorable tournaments with ten colors as a subroutine.

▶ **Lemma 14.** *A 3-colorable tournament can be colored with $O(\sqrt{n})$ colors in polynomial time.*

**Proof.** Let $T = (V, A)$ be a 3-colorable tournament. Notice that $T$ has at least three vertices each of whose out-neighborhoods is 2-colorable. To see this, consider any proper 3-coloring of $T$. Each color spans a transitive subtournament and each transitive subtournament has a sink vertex that has outgoing arcs only towards the other two colors.

For any vertex, if its out-neighborhood is 2-colorable, we can color its out-neighborhood with 10 colors by Theorem 7. So we can try to run the algorithm for the out-neighborhood of every vertex, and the algorithm will successfully produce a 10-coloring of the out-neighborhood of at least three vertices.

Therefore, if the minimum outdegree is at least $\sqrt{n}$, we find a transitive set of size at least $\sqrt{n}/10$. On the other hand, if the minimum outdegree is smaller than $\sqrt{n}$, we will make progress another way. In this case, let $u$ be a vertex with outdegree smaller than $\sqrt{n}$. Then, we add $u$ to a set $S$, and continue the algorithm on the subtournament of $T$ induced on $V \setminus N^+[u]$. We continue this until we find a transitive subtournament of size at least $\sqrt{n}/20$ or until we have removed half the vertices. In the first case, we will have found a transitive set of size $\Omega(\sqrt{n})$, and in the second case, the set $S$ will be transitive, and also of size $\Omega(\sqrt{n})$.

In conclusion, since we can find a transitive set of size $\Omega(\sqrt{n})$ in polynomial time, we can repeat the procedure recursively to find a coloring with $O(\sqrt{n})$ colors in polynomial time (see [5] for example). ◀

We can also use the decomposition of Section 2 to get a coloring with fewer colors based on a reduction to coloring 3-colorable graphs.

▶ **Theorem 15.** *If we can efficiently color a 3-colorable graph $G$ with $k$ colors, then we can efficiently color a 3-colorable tournament with $50k$ colors.*

**Proof.** Let $T = (V, A)$ be a 3-colorable tournament. For every arc $e \in A$, try coloring $N(e)$ with 10 colors using Theorem 7. If the algorithm fails, the neighborhood of the edge is not 2-colorable, and thus the edge is not monochromatic in any 3-coloring. Let $F \subset E$ denote the set of arcs whose neighborhoods cannot be colored with 10 colors using our algorithm. Ignore the direction of the arcs in $F$ and consider the graph $G = (V, F)$. This graph must be 3-colorable, since no arc in $F$ is monochromatic in any 3-coloring of $T$.

Now let us show that from a coloring of $G$ with $k$ colors, we can obtain a coloring of $T$ with $50k$ colors. Consider a coloring of the graph $G = (V, F)$ and let $V_i$ be the vertices colored with color $i$ in this coloring. Consider the induced subtournament $T' = T[V_i]$; it has no arc in $F$ and thus the neighborhood of every arc in this tournament can be colored efficiently with 10 colors. Furthermore, by Lemma 12 and Theorem 7, there are vertices $u$ and $v$ in $T'$ such that $N^+_{T'}(u) \cup N^-_{T'}(v)$ is efficiently 10-colorable. So by Lemma 5, we can efficiently color $T'$ with 50 colors. We can do this for the subtournament $T[V_i]$ for each of the $i$ colors used to color $G$. ◀

Combining this Lemma with approximation algorithm [24], which colors a 3-colorable graph with fewer than $n^{\frac{1}{5}}$ colors, we obtain the same asymptotic bound for 3-colorable tournaments.

▶ **Corollary 16.** *Let $T$ be a 3-colorable tournament on $n$ vertices. Then, $\vec{\chi}_{\mathcal{C}}(T) \leq O(n^{0.19996})$.*

We can extend Theorem 15 to a more general case.

▶ **Lemma 17.** *Let $f$ and $g$ be functions such that we can efficiently color $k$-colorable graphs (respectively, $k$-colorable tournaments) with $g(k)$ (respectively, $f(k)$) colors. Then $f(k) \leq 5 \cdot f(k-1) \cdot g(k)$.*

**Proof.** We use the same reduction as in the proof of Theorem 15, but now $F$ is the set of arcs whose neighborhoods cannot be efficiently $f(k-1)$-colored. Then each $V_i$ in $G$ is colored with $5 \cdot f(k-1)$ colors. So we need a total of $5 \cdot f(k-1) \cdot g(k)$ colors. ◀

## 3.3 Light Tournaments

Our goal in this section is to prove the following theorem.

▶ **Theorem 18.** *Let $T$ be a light tournament. Then $\vec{\chi}_{\mathcal{C}}(T) \leq 9$.*

We will prove Theorem 18 by showing that every light tournament has a $c$-vertex chain for some constant $c$. To do this, we will find one vertex whose in-neighborhood we can color efficiently with a constant number of colors, and another whose out-neighborhood we can color efficiently with a constant number of colors. We will start by establishing some structural claims about light tournaments which are adapted from [4].

Throughout this section $T = (V, A)$ will denote a light tournament. Note that we do not assume that $T$ is necessarily 2-colorable. Recall that a $C_3$ is a directed triangle.

▶ **Definition 19.** *Define a* $C_3$-chain *of length $\ell$ in $T$ to be a set of $\ell$ vertex disjoint $C_3$'s, $X = (X_1, X_2, X_3, \ldots, X_\ell)$, such that for each $i \in \{1, \ldots, \ell - 1\}$, $X_i \Rightarrow X_{i+1}$.*

A *backwards arc* in a $C_3$-chain is an arc $uv$ with $u \in X_i$ and $v \in X_j$ for $j < i$.

▶ **Lemma 20.** *A $C_3$-chain has no backwards arcs.*

This follows from the following claim.

▷ Claim 21. If $X = (X_1, X_2, \ldots, X_\ell)$ is a $C_3$-chain of length $\ell$, then $X_i \Rightarrow X_j$ for $i < j$, where $1 \leq i < j \leq \ell$.

Proof. Notice that there are no arcs from $X_{i+1}$ to $X_i$, since by definition of a $C_3$-chain, we have all arcs from $X_i$ to $X_{i+1}$. Moreover, there is no arc $uv$ from $X_{i+2}$ to $X_i$ since otherwise triangle $X_{i+1}$ would appear in the neighborhood $N(uv)$, meaning that $uv$ is heavy, which is a contradiction. This implies that all arcs go from $X_i$ to $X_{i+2}$ (since $T$ is a tournament). Now suppose $j > i + 2$. If there is a back arc $uv$ from $u \in X_j$ to $v \in X_i$, then $uv$ is a heavy arc, because $X_{j-1}$ would be in $N(uv)$ since by induction we have all arcs from $X_i$ to $X_{j-1}$ and from $X_{j-1}$ to $X_j$. ◁

Let us fix $X = (X_1, X_2, \ldots, X_\ell)$ to be a $C_3$-chain in $T$, and let $W = V(T) \setminus V(X)$. Initially, $X$ can be of any length $\ell \geq 1$.

▷ Claim 22. For $w \in W$:
1. If $w \Rightarrow X_i$, then $w \Rightarrow X_j$ for all $j \geq i$.
2. If $X_i \Rightarrow w$, then $X_j \Rightarrow w$ for all $j \leq i$.

Proof. Suppose $w \Rightarrow X_i$ and there is an arc $uw$ with $u \in X_j$ for $j > i$. Then $uw$ is a heavy arc. Similarly, suppose $X_i \Rightarrow w$ and there is an arc $wu$ with $u \in X_j$ for $j < i$, then $wu$ is a heavy arc. ◁

We partition the vertices in $W$ into zones $(Z_0, Z_1, \ldots, Z_\ell)$ using the following criteria. For $w \in W$, if $i$ is the highest index such that $X_i \Rightarrow w$, then $w$ is assigned to zone $Z_i$. If there is no such $X_i$, then $w$ is assigned to zone $Z_0$.

Say a vertex $w \in W$ is *clear* if $w \Rightarrow X_i$ or $X_i \Rightarrow w$ for all $X_i$ in $H$. Let $C \subseteq W$ be the set of clear vertices.

▷ Claim 23. If $C$ is not transitive, we can extend $X$.

Proof. If the set $Z_i \cap C$ contains a triangle, then we can extend $X$ by adding a new triangle to the chain between $X_i$ and $X_{i+1}$.

If there is no $i$ such that $Z_i \cap C$ contains a triangle, then we claim that $C$ is transitive. This follows from the observation that there are no backwards arcs from $Z_j \cap C$ to $Z_i \cap C$ for $i < j$. Indeed, should such an arc $uv$ from $Z_j \cap C$ to $Z_i \cap C$ exist, then $X_{i+1} \subset N(uv)$, so $uv$ would be heavy. ◁

We say that $X$ is a *maximal $C_3$-chain* if $C$ is transitive. Let us also now define the *unclear* vertices $U$, where $U = W \setminus C$. In a maximal $C_3$-chain $X = (X_1, \ldots, X_\ell)$, notice that for a vertex $a \in X_1$, we have $N^-(a) \cap U \subseteq N^\pm(X_1)$. (This is because if a vertex $u \in N^-(a)$ has $u \Rightarrow X_i$, then $u$ would be a clear vertex.)

▷ Claim 24. We can efficiently find two directed triangles $X_1 = abc$ and $X_\ell = xyz$ such that the set $S = \{v \mid v \Rightarrow X_1 \text{ or } X_\ell \Rightarrow v\}$ is transitive.

Proof. Find a maximal $C_3$-chain $X$ and let $\ell$ be the length of this chain. Let $abc = X_1$ and $xyz = X_\ell$. The set of vertices $\{v \mid v \Rightarrow X_1 \text{ or } X_\ell \Rightarrow v\}$ is a subset of $C$ and is therefore transitive. ◁

▷ Claim 25. Let $xyz$ be a directed triangle. Then $\vec{\chi}_{\mathcal{C}}(N^\pm(\{x, y, z\})) \leq 3$.

Proof. Each vertex $v \in N^\pm(\{x, y, z\})$ belongs to $N(xy), N(yz)$ or $N(zx)$. Since each of these sets is transitive, we conclude that $N^\pm(\{x, y, z\})$ can be colored with three colors. ◁

We can now easily prove Theorem 18, which is a corollary of Lemma 6 and the following lemma.

▶ **Lemma 26.** *Let $T$ be a light tournament. Then $T$ has a 5-vertex chain.*

**Proof.** Recall that for a vertex $a \in X_1$, we have $N^-(a) \cap U \subseteq N^\pm(X_1)$. If $X_1 = abc$, notice that for $v \in N^-(a) \cap U$, $v \notin N(ca)$. Thus, $N^-(a) \cap U \subseteq N(ab) \cup N(bc)$, which is efficiently 2-colorable. Making an analogous argument for $N^+(z) \cap U$, we conclude that $(N^+(z) \cup N^-(a)) \cap U$ is efficiently 4-colorable. The rest of the vertices in $N^+(z) \cup N^-(a)$ belong to the set $S$ defined in Claim 24 and can be colored with one color. Therefore $\vec{\chi}_{\mathcal{C}}(N^+(z) \cup N^-(a)) \leq 5$, so we can use $z$ and $a$ as the endpoints of a 5-vertex chain. ◀

The approach in this section can be extended to bound the chromatic number of a more general subclass of heroes. See the full version [28] for details.

# 4 Hardness of Approximate Coloring in Tournaments

In this section, we examine the hardness of approximate coloring of tournaments. [8] showed that deciding if a tournament can be 2-colored is NP-hard. Later, [15] proved that for any $k$, it is NP-hard to decide if a tournament is $k$-colorable.

We will first improve upon these NP-hardness results and then show hardness of coloring $k$-colorable tournaments for $k \geq 3$ with $O(1)$ colors under the $d$-To-1 conjecture. The $d$-To-1 conjecture was first introduced by Khot alongside the famous Unique Games conjecture [26], and has since been used to show hardness of coloring 3-colorable graphs with $O(1)$ colors [18].

First notice that the search problem must be at least as hard as its decisional equivalent.

▶ **Observation 27.** *Let $k < \ell$ be any two constants. If we can color $k$-colorable tournaments with $\ell$ colors, then we can distinguish $k$-colorable tournaments from tournaments with chromatic number at least $\ell + 1$.*

This comes immediately from the fact that if we could $\ell$-color all $k$-colorable tournaments, then we could see that they do not have chromatic number $\ell + 1$ or greater. The hardness of distinguishing between chromatic number $k$ and greater or equal to $\ell + 1$ is therefore commonly established as a way of implying the hardness of coloring $k$-colorable graphs with $\ell$ colors (see for example [6]).

All proofs of the theorems in this section are provided in the full version [28].

## 4.1 NP-Hardness of Approximate Coloring of $k$-Colorable Tournaments

It was shown previously that it is NP-hard to color a 2-colorable tournament with 2 colors [8, 15]. We prove a stronger theorem, that it is NP-hard to 3-color a 2-colorable tournament.

▶ **Theorem 28.** *It is NP-hard, given a tournament $T$, to distinguish whether $\vec{\chi}(T) = 2$ or $\vec{\chi}(T) \geq 4$.*

The proof of this Theorem relies on a reduction from the problem of coloring 2-colorable tournaments with three colors to the problem of coloring 2-colorable 3-uniform hypergraphs with six colors. This problem is NP-hard, since it was proven that coloring 2-colorable 3-uniform hypergraphs with any constant number of colors is NP-hard [11].

We then use a recursive construction that starts with the tournament obtained in the proof of Theorem 28 to generalize the hardness of approximation to $k$-colorable tournaments for any constant $k$.

▶ **Theorem 29.** *It is NP-hard, given a tournament $T$ and a constant $k$, to distinguish whether $\vec{\chi}(T) = k$ or $\vec{\chi}(T) \geq 2k$.*

## 4.2 Reduction from Coloring Graphs to Coloring Tournaments

In Section 3.2, we showed that if we can color a 3-colorable graph with $k$ colors, then we can color a 3-colorable tournament with $50k$ colors. We give a reduction in the other direction: We show that the problem of coloring a $k$-colorable graph with $\ell$ colors is reducible to the problem of coloring a $k$-colorable tournament with $\ell$ colors.

▶ **Theorem 30.** *Given any two constants $k, \ell \geq 3$, if we can efficiently distinguish $k$-colorable tournaments and tournaments with chromatic number at least $\ell$, then we can efficiently distinguish $k$-colorable graphs and graphs with chromatic number at least $\ell$.*

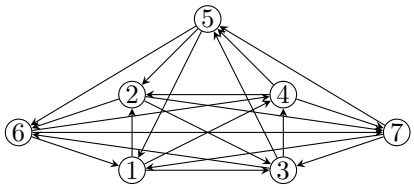



**Figure 2** 3-chromatic light tournament.

A corollary of this reduction is the hardness of coloring tournaments under the $d$-To-1 Conjecture of Khot [26]; [18] showed that assuming the $d$-To-1 Conjecture, it is hard to color 3-colorable graphs with $O(1)$ colors, and using our reduction, we can extend this hardness to tournaments.

▶ **Corollary 31.** *Let $3 \leq k < \ell$ be any two constants. Then if the $d$-To-1 conjecture is true, we cannot distinguish between tournaments with chromatic number $k$ and tournaments with chromatic number at least $\ell$.*

## 4.3 Hardness of Approximation for General Tournaments

Coloring digon-free digraphs has been shown to be NP-hard to approximate within a factor of $n^{1/2-\epsilon}$ [14]. This proof can easily be extended to the case of tournaments, which provides the following theorem.

▶ **Theorem 32.** *Given any arbitrarily small constant $\epsilon > 0$, it is NP-hard to approximate the chromatic number of tournaments within a factor of $n^{1/2-\epsilon}$.*

# 5 Conclusion

There are many open questions related to the theorems we have presented since all the rows in Table 2 present gaps between the upper and lower bounds. One example is light tournaments: What is the maximum number of colors required to color a light tournament? From Theorem 18, we know that light tournaments have dichromatic number at most 9. On the other hand, there exist light tournaments that are not 2-colorable. An example of such a tournament is the Paley tournament $P_7$, one of the four 3-chromatic tournaments on seven vertices [33]. This tournament is represented in Figure 2. We have not found any light tournament with chromatic number at least four. The Paley tournament $P_{11}$ is the unique 4-chromatic tournament on 11 vertices [33]. A light 4-chromatic tournament would have to have at least 13 vertices as [3] proved that any 4-chromatic tournament on 12 vertices must contain an induced copy of $P_{11}$ and $P_{11}$ is not light.

Moreover, notice that if we could show that it is hard to color a 2-colorable tournament with four colors (rather than three as per Theorem 28), this would imply hardness of coloring a 2-colorable light tournament with two colors by Observation 8. Indeed, we have no hardness results for coloring light tournaments. Any upper bound of $c$ on their dichromatic number would imply that it cannot be NP-hard to color them with $c$ colors, because the property of being light is checkable in polynomial time (unlike the property of being, say, 2-colorable).

Another observation is the relation of coloring tournaments and the feedback vertex set (FVS) problem on tournaments. There is an elegant 2-approximation for this problem [30]. Notice that Theorem 7 implies that in a 2-colorable tournament, we can efficiently find a FVS of size at most $9n/10$. In contrast, the algorithm in [30] could just return the whole vertex set

if the two transitive sets were of roughly equal size. Finally, we mention that, analogous to a well-studied question for general graphs [10, 27], one can ask what is the largest transitive induced subtournament that one can efficiently find in a 2-colorable tournament? Is it larger than $n/10$?

Finally, we remark that an implication of Theorem 15 is that proving any hardness of coloring 3-colorable tournaments would then provide hardness of coloring 3-colorable graphs with 50 times fewer colors. Since it has taken around 20 years to go from proving NP-hardness of coloring a 3-colorable graph with four colors [25, 16, 17] to NP-hardness of coloring a 3-colorable graph with five colors [6], it would be interesting to see if we can prove hardness of coloring 3-colorable tournaments for a constant larger than five (at least five is shown in Theorem 29), or perhaps show that the two problems are actually equivalent.

## References


1 Noga Alon, Pierre Kelsen, Sanjeev Mahajan, and Hariharan Ramesh. Coloring 2-colorable hypergraphs with a sublinear number of colors. *Nordic Journal of Computing*, 3:425–439, 1996.

2 Noga Alon, János Pach, and József Solymosi. Ramsey-type theorems with forbidden subgraphs. *Combinatorica*, 21(2):155–170, 2001.

3 Thomas Bellitto, Nicolas Bousquet, Adam Kabela, and Théo Pierron. The smallest 5-chromatic tournament. *arXiv*, 2022. `arXiv:2210.09936`.

4 Eli Berger, Krzysztof Choromanski, Maria Chudnovsky, Jacob Fox, Martin Loebl, Alex Scott, Paul Seymour, and Stéphan Thomassé. Tournaments and colouring. *Journal of Combinatorial Theory, Series B*, 103(1):1–20, 2013.

5 Avrim Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41(3):470–516, 1994.

6 Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 602–613, 2019.

7 Hui Chen and Alan Frieze. Coloring bipartite hypergraphs. In *Fifth International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 345–358, 1996.

8 Xujin Chen, Xiaodong Hu, and Wenan Zang. A min-max theorem on tournaments. *SIAM Journal on Computing*, 37(3):923–937, 2007.

9 Maria Chudnovsky. The Erdös-Hajnal Conjecture – A survey. *Journal of Graph Theory*, 75(2):178–190, 2014.

10 Irit Dinur, Subhash Khot, Will Perkins, and Muli Safra. Hardness of finding independent sets in almost 3-colorable graphs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 212–221, 2010.

11 Irit Dinur, Oded Regev, and Clifford Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005.

12 Paul Erdős and András Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1-2):37–52, 1989.

13 Paul Erdos and Leo Moser. On the representation of directed graphs as unions of orderings. *Math. Inst. Hung. Acad. Sci*, 9:125–132, 1964.

14 Tomás Feder, Pavol Hell, and Carlos Subi. Complexity of acyclic colorings of graphs and digraphs with degree and girth constraints. *arXiv*, 2019. `arXiv:1907.00061`.

15 Jacob Fox, Lior Gishboliner, Asaf Shapira, and Raphael Yuster. The removal lemma for tournaments. *Journal of Combinatorial Theory, Series B*, 136:110–134, 2019.

16 Venkatesan Guruswami and Sanjeev Khanna. On the hardness of 4-coloring a 3-colorable graph. In *Proceedings 15th Annual IEEE Conference on Computational Complexity (CCC)*, pages 188–197, 2000.

17 Venkatesan Guruswami and Sanjeev Khanna. On the hardness of 4-coloring a 3-colorable graph. *SIAM Journal on Discrete Mathematics*, 18(1):30–40, 2004.

**18** Venkatesan Guruswami and Sai Sandeep. $d$-To-1 hardness of coloring 3-colorable graphs with $O(1)$ colors. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020.

**19** Magnús M Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993.

**20** Ararat Harutyunyan, Tien-Nam Le, Alantha Newman, and Stéphan Thomassé. Coloring dense digraphs. *Combinatorica*, 39(5):1021–1053, 2019.

**21** Ararat Harutyunyan, Tien-Nam Le, Stéphan Thomassé, and Hehui Wu. Coloring tournaments: From local to global. *Journal of Combinatorial Theory, Series B*, 138:166–171, 2019.

**22** Johan Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

**23** David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998.

**24** Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *Journal of the ACM*, 64(1):1–23, 2017.

**25** Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.

**26** Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 767–775, 2002.

**27** Subhash Khot and Rishi Saket. Hardness of finding independent sets in 2-colorable and almost 2-colorable hypergraphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1607–1625, 2014.

**28** Felix Klingelhoefer and Alantha Newman. Coloring tournaments with few colors: Algorithms and complexity. *arXiv*, 2023. `arXiv:2305.02922`.

**29** Michael Krivelevich, Ram Nathaniel, and Benny Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. *Journal of Algorithms*, 41(1):99–113, 2001.

**30** Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-Approximating feedback vertex set in tournaments. *ACM Transactions on Algorithms*, 17(2):1–14, 2021.

**31** László Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12, 1973.

**32** Victor Neumann-Lara. The dichromatic number of a digraph. *Journal of Combinatorial Theory, Series B*, 33(3):265–270, 1982.

**33** Victor Neumann-Lara. The 3 and 4-dichromatic tournaments of minimum order. *Discrete Mathematics*, 135(1-3):233–243, 1994.

**34** Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, 1983.

**35** David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 681–690, 2006.

# Bellman–Ford Is Optimal for Shortest Hop-Bounded Paths

**Tomasz Kociumaka**
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

**Adam Polak**
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

## Abstract

This paper is about the problem of finding a shortest $s$-$t$ path using at most $h$ edges in edge-weighted graphs. The Bellman–Ford algorithm solves this problem in $O(hm)$ time, where $m$ is the number of edges. We show that this running time is optimal, up to subpolynomial factors, under popular fine-grained complexity assumptions.

More specifically, we show that under the APSP Hypothesis the problem cannot be solved faster already in undirected graphs with nonnegative edge weights. This lower bound holds even restricted to graphs of arbitrary density and for arbitrary $h \in O(\sqrt{m})$. Moreover, under a stronger assumption, namely the Min-Plus Convolution Hypothesis, we can eliminate the restriction $h \in O(\sqrt{m})$. In other words, the $O(hm)$ bound is tight for the entire space of parameters $h$, $m$, and $n$, where $n$ is the number of nodes.

Our lower bounds can be contrasted with the recent near-linear time algorithm for the negative-weight Single-Source Shortest Paths problem, which is the textbook application of the Bellman–Ford algorithm.

**2012 ACM Subject Classification** Theory of computation → Shortest paths

**Keywords and phrases** Fine-grained complexity, graph algorithms, lower bounds, shortest paths

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.72

**Funding** *Adam Polak*: Part of this work was done at École Polytechnique Fédérale de Lausanne, supported by the Swiss National Science Foundation projects *Lattice Algorithms and Integer Programming* (185030) and *Complexity of Integer Programming* (CRFS-2_207365).

**Acknowledgements** The second author would like to thank Danupon Nanongkai and Luca Trevisan for bringing his attention to the problem discussed in this paper, Alexandra Lassota – for useful feedback on an early draft of the manuscript, and Imbir – a ginger tabby, who supervised initial stages of this work.

## 1 Introduction

The Bellman–Ford algorithm [24, 14, 4] is the textbook solution for the Single-Source Shortest Paths (SSSP) problem in graphs with negative edge weights. It runs in $O(nm)$ time, where $n$ denotes the number of nodes and $m$ is the number of edges. If we limit the outer for-loop (see Algorithm 1) to only $h \leqslant n-1$ iterations, the algorithm computes single-source shortest paths that use at most $h$ edges (or *hops*) and runs in $O(hm)$ time.

**Algorithm 1** The Bellman–Ford algorithm.

$d^{(0)} \leftarrow [+\infty, +\infty, \ldots, +\infty]$;
$d^{(0)}[s] \leftarrow 0$;
**for** $i$ **from** 1 **to** $n-1$ **do**
  $d^{(i)} \leftarrow d^{(i-1)}$;
  **foreach** edge $(u, v) \in E$ **do**
    $d^{(i)}[v] \leftarrow \min\{d^{(i)}[v], d^{(i-1)}[u] + w(u, v)\}$;


Negative-weight SSSP has seen a lot of improvements over Bellman–Ford's running time: scaling algorithms [15, 16, 18], which eventually led to $O(n^{1/2}m \log W)$ running time, where $W$ denotes the maximum absolute value of a negative edge weight; interior-point methods for the more general Minimum-Cost Flow problem, which recently led to an almost-linear $O(m^{1+o(1)} \log W)$ time algorithm [8]; and finally, the recent combinatorial near-linear $O(m \log^8(n) \log W)$ time algorithm [5], subsequently improved to run in $O(m \log^2(n) \log(nW) \log \log n)$ time [7].

*Can we get similar improvements for the problem of finding shortest hop-bounded paths?*

This basic question stays embarrassingly open. Even in undirected graphs with only nonnegative edge weights, Bellman–Ford remains the fastest known algorithm for that problem. In this paper, we give a negative answer to the above question, up to subpolynomial factors, under popular fine-grained complexity assumptions.

## 1.1 Our results

Let us begin with formally stating the computational problem that we study: Given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Z}$, two distinguished nodes $s, t \in V$, and a nonnegative integer $h \in \mathbb{Z}_{\geqslant 0}$, find (the length of) a shortest path from $s$ to $t$ that uses at most[1] $h$ edges. We call such paths *$h$-hop-bounded*, or simply *hop-bounded* when $h$ is implicit in the context.

In this paper, we give two fine-grained reductions, each proving that the $O(hm)$ running time of the Bellman–Ford algorithm is conditionally optimal for the problem, up to subpolynomial factors. Our two hardness results differ from each other in (1) how the parameters $n, m, h$ of the hard instances relate to each other, and (2) which hardness assumption is required. Table 1 summarizes these differences.

Our first result holds under the APSP Hypothesis.

▶ **Theorem 1.** *Unless the APSP Hypothesis fails, there is neither an $O(h^{1-\varepsilon}m)$ nor an $O(hm^{1-\varepsilon})$ time algorithm for finding the length of a shortest $h$-hop-bounded $s$-$t$ path in undirected graphs with nonnegative edge weights, for any constant $\varepsilon > 0$.*

*This holds even restricted to instances with density $n = \Theta(\sqrt{m})$ and hop bound $h = \Theta(m^\eta)$ for arbitrarily chosen $\eta \in (0, 1/2]$.*

Although the hard instances in Theorem 1 are dense, one can trivially obtain sparser instances by adding isolated nodes. Indeed, such nodes influence neither the length of a shortest $h$-hop-bounded $s$-$t$ path nor the running time bounds as functions of $h$ and $m$.

▶ **Corollary 2.** *The result of Theorem 1 holds even restricted to instances with density $n = \Theta(m^\nu)$ and hop bound $h = \Theta(m^\eta)$ for arbitrarily chosen $\nu \in [1/2, 1]$ and $\eta \in (0, 1/2]$.*

We remark that it is not very surprising that our reduction from APSP can only produce instances with $h \in O(\sqrt{m})$. The conjectured time complexity of APSP in $n$-node graphs is $n^{3-o(1)} = |\mathsf{input}|^{3/2-o(1)}$. For $h = \Theta(m^\eta)$, the $O(hm)$ time bound is actually $O(|\mathsf{input}|^{1+\eta})$. Fine-grained reductions from problems with smaller complexity to problems with larger complexity are possible (see, e.g., [22]) but rare, and to our best knowledge no such reduction from APSP is known. If Theorem 1 worked for $\eta > 1/2$, this would be the first such example.

[1] We could also consider a variant of the problem asking for a *walk* with *exactly* $h$ edges. It is the harder of the two variants (adding a length-0 self-loop to node $s$ reduces the "at most $h$" variant to the "exactly $h$" variant), and we prove the hardness of the easier one already.

**Table 1** Summary and comparison of our conditional hardness results for the problem of finding (the length of) a shortest hop-bounded path between two nodes.

| | Density | Hops | Hypothesis |
|---|---|---|---|
| Corollary 2 | $n = \Theta(m^\nu)$, $\nu \in [1/2, 1]$ | $h = \Theta(m^\eta)$, $\eta \in (0, 1/2]$ | APSP |
| Corollary 4 | $n = \Theta(m^\nu)$, $\nu \in [1/2, 1]$ | $h = \Theta(m^\eta)$, $\eta \in [1/2, \nu]$ | Min-Plus Convolution |

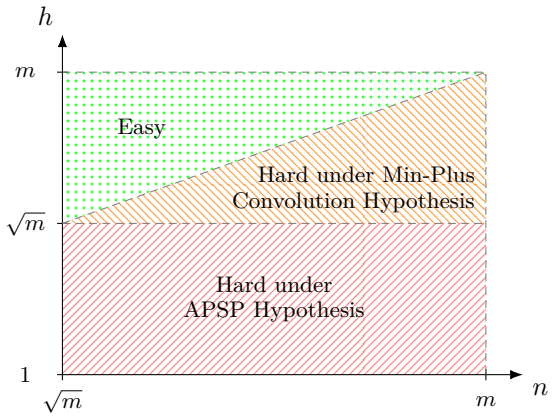



**Figure 1** Parameter space. The upper-left triangle represents the case of $h \geqslant n$, where the problem degenerates to the standard Shortest Path problem, without hop bound, which can be solved in $\tilde{O}(m)$ time.

In order to cover the remaining combinations of parameters, we use a stronger hypothesis, concerning a problem with conjectured quadratic time complexity, namely the Min-Plus Convolution Hypothesis. Since this hypothesis implies the APSP Hypothesis, it is also a sufficient condition for Theorem 1 and thus gives hardness for the entire parameter space.

▶ **Theorem 3.** *Unless the Min-Plus Convolution Hypothesis fails, there is neither an $O(h^{1-\varepsilon}m)$ nor an $O(hm^{1-\varepsilon})$ time algorithm for finding the length of a shortest $h$-hop-bounded $s$-$t$ path problem in undirected graphs with nonnegative edge weights, for any constant $\varepsilon > 0$.*

*This holds even restricted to instances with density $n = \Theta(m^\eta)$ and hop bound $h = \Theta(m^\eta)$ for arbitrarily chosen $\eta \in [1/2, 1]$.*

Just like before, we can sparsify the hard instances by adding isolated nodes.

▶ **Corollary 4.** *The result of Theorem 3 holds even restricted to instances with density $n = \Theta(m^\nu)$ and hop bound $h = \Theta(m^\eta)$ for arbitrarily chosen $\nu \in [1/2, 1]$ and $\eta \in [1/2, \nu]$.*

Combining Corollaries 2 and 4, we cover the entire range of parameters $\nu \in [1/2, 1]$ and $\eta \in (0, \nu]$ for which the $O(hm)$ running time is optimal; see Figure 1.

Let us point out that, even though the Bellman–Ford algorithm finds paths from a single source $s$ to all the nodes in the graph, the above two hardness results hold even for the easier problem of finding a single path between two distinguished nodes $s$ and $t$. Moreover, note that any (shortest path) algorithm for directed graphs could also be used for undirected graphs (but not necessarily vice versa). Bellman–Ford works in directed graphs with possibly negative edge weights, while our hardness results already hold for undirected graphs with nonnegative edge weights.

## 1.2 Hardness assumptions

In this section, we briefly recall the two hypotheses that we use in our theorems.

The APSP Hypothesis is the assertion that the All-Pairs Shortest Paths (APSP) problem in $n$-node graphs cannot be solved in truly subcubic $O(n^{3-\varepsilon})$ time for any constant $\varepsilon > 0$. It is one of the three main hypotheses in fine-grained complexity, the other two being the 3SUM Hypothesis and Strong Exponential Time Hypothesis (SETH) [26]. The APSP Hypothesis is strengthened by the existence of a large class of problems that are equivalent to the APSP problem under subcubic reductions [27, 26], and by the lack of a truly subcubic algorithm for any of these problems.

In the Min-Plus Convolution problem, we are given two sequences $(a[i])_{i=0}^{n-1}$, $(b[i])_{i=0}^{n-1}$, and the goal is to output sequence $(c[i])_{i=0}^{n-1}$ defined as $c[k] = \min_{i+j=k}(a[i] + b[j])$. So far, only subpolynomial $2^{O(\sqrt{\log n})}$-factor improvements [6, 28] over the naive quadratic running time are known. The Min-Plus Convolution Hypothesis [21, 11] states that the problem cannot be solved in truly subquadratic time, that is, $O(n^{2-\varepsilon})$ for any constant $\varepsilon > 0$. Similarly to APSP, there is also a class (albeit smaller) of problems equivalent to Min-Plus Convolution under subquadratic reductions [11].[2]

The two hypotheses are closely related because APSP is runtime-equivalent (up to constant factors) to the Min-Plus Product problem [13], which is the matrix product analogue of Min-Plus Convolution. There is a reduction from the convolution to the product problem [6], which entails that the Min-Plus Convolution Hypothesis implies the APSP Hypothesis, and the former is therefore a stronger assumption.

As is customary in fine-grained complexity, these hypotheses, as well as all the results in this paper, are stated in the word RAM model of computation with $O(\log n)$-bit machine words. We assume all input numbers fit into single machine words. Alternatively, the APSP Hypothesis is sometimes stated as follows [26]: For every $\varepsilon > 0$ there is a constant $c$ such that APSP cannot be solved in $O(N^{3-\varepsilon})$ time in $N$-node graphs with edge weights in $\{-N^c, \ldots, N^c\}$. Our results could also be stated this way because our reductions do not increase weights by more than polynomial factors.

## 1.3 Related work

Hop-bounded paths are studied in various areas related to graph algorithms, e.g., distributed algorithms [17], dynamic algorithms [25, 23], or even polyhedral combinatorics [12]. Problems of finding shortest hop-bounded paths appear, e.g., in the context of quality-of-service (QoS) routing in networks [2, 9].

Guérin and Orda [19] and Cheng and Ansari [10] studied the problem of finding shortest $h$-hop-bounded paths from single source $s$ to all nodes in the graph and for all hop bounds $h \leqslant H$. They proved an $\Omega(Hm)$ lower bound for that problem against so-called *path-comparison-based algorithms*, i.e., algorithms that only access the edge weights by comparing the lengths of two paths. Although Dijkstra and Bellman–Ford are known to be path-comparison-based, algebraic algorithms relying on fast matrix multiplication are not.

[2] One of the problems equivalent to Min-Plus Convolution is the Knapsack problem on instances with target value $t = \Theta(n)$. Recall that Knapsack can be solved in $O(nt)$ time by a dynamic programming algorithm due to Bellman [3]. Hence, we can half-jokingly rephrase Theorem 3 and say that if one Bellman's algorithm is optimal for Knapsack, then the other Bellman's algorithm is optimal for shortest hop-bounded paths.

**Bicriteria Path.** In the Bicriteria Path problem, we are given a graph $G = (V, E)$ with two types of nonnegative edge weights $l, c : E \to \mathbb{Z}$, called *lengths* and *costs*, respectively; two *budgets* $L, C \in \mathbb{Z}$; and two distinguished nodes $s, t \in V$. The goal is to find a path from $s$ to $t$ with the total length at most $L$ and the total cost at most $C$. Joksch's algorithm [20] solves the problem in pseudopolynomial $O(\min(L, C) \cdot m)$ time. For the special case of all edge costs equal to 1, the Bicriteria Path problem is equivalent to the problem we study in this paper and, moreover, Joksch's algorithm runs in the same time as the Bellman–Ford algorithm.

Abboud, Bringmann, Hermelin, and Shabtay [1] proved that, unless SETH fails, there is no algorithm solving the Bicriteria Path problem on sparse graphs (with $m = \Theta(n)$ edges) with budgets $L, C = \Theta(n^\gamma)$ in time $O(n^{1+\gamma-\varepsilon})$ for any $\varepsilon > 0$ and $\gamma > 0$. In other words, they proved that Joksch's algorithm is conditionally optimal, up to subpolynomial factors. Their reduction, however, heavily uses both types of edge weights, and hence it does not imply any lower bound for the special case with unit costs, i.e., for our problem of interest.

## 2 Hardness under APSP Hypothesis

**Preliminaries.** Given a complete tripartite graph $G = (A \cup B \cup C, E)$ with edge weights $w : E \to \mathbb{Z}$, the Negative Triangle problem asks to find three nodes $a \in A$, $b \in B$, and $c \in C$ with $w(a, b) + w(b, c) + w(c, a) < 0$. Vassilevska Williams and Williams [27] proved that APSP and Negative Triangle are equivalent under subcubic reductions. In particular, unless the APSP Hypothesis fails, there is no $O(N^{3-\varepsilon})$ time algorithm for Negative Triangle with $|A| = |B| = |C| = N$, for any $\varepsilon > 0$.

Via a by now standard argument, under the same assumption, for any $\varepsilon > 0$, there is no $O(N^{\alpha+2-\varepsilon})$ time algorithm for the problem restricted to instances with $|A| = \Theta(N^\alpha)$ and $|B| = |C| = N$ for arbitrarily chosen $\alpha \in (0, 1]$. Specifically, the reduction partitions the original set $A$ into $\Theta(N^{1-\alpha})$ subsets of size $\Theta(N^\alpha)$ each; the sets $B$ and $C$ are copied to all $\Theta(N^{1-\alpha})$ produced instances. A negative triangle exists in the original instance if and only if it exists in at least one of the produced instances. Thus, if each of the obtained instances could be solved in $O(N^{\alpha+2-\varepsilon})$ time, then the original instance could be solved in $O(N^{1-\alpha} \cdot (N^2 + N^{\alpha+2-\varepsilon})) = O(N^{3-\alpha} + N^{3-\varepsilon})$ time[3], violating the APSP Hypothesis.

**Reduction.** We show how to reduce an instance of Negative Triangle, with $|A| = \Theta(N^\alpha)$ and $|B| = |C| = N$, to finding the minimum length of an $h$-hop-bounded $s$-$t$ path in an undirected graph with $n = \Theta(N)$ nodes, $m = \Theta(N^2)$ edges, and the hop bound $h = \Theta(N^\alpha)$. In order to prove Theorem 1, we set $\alpha = 2\eta$ so that $n = \Theta(N) = \Theta(\sqrt{m})$ and $h = \Theta(N^\alpha) = \Theta(N^{2\eta}) = \Theta(m^\eta)$. An $O(h^{1-\varepsilon}m)$ time (or $O(hm^{1-\varepsilon})$ time) algorithm finding a shortest $h$-hop-bounded $s$-$t$ path would thus yield an $O(N^{\alpha(1-\varepsilon)+2}) = O(N^{\alpha+2-\alpha\varepsilon})$ time (respectively, $O(N^{\alpha+2-2\varepsilon})$ time) algorithm for the original instance of the Negative Triangle problem. As argued above, no such algorithm exists unless the APSP Hypothesis fails.

Let $P = |A|$. Suppose that $A = \{a_1, \ldots, a_P\}$, $B = \{b_1, \ldots, b_N\}$, and $C = \{c_1, \ldots, c_N\}$. Moreover, let $W$ denote the maximum absolute value of an edge weight. Create an undirected graph (see Figure 2) with node set $A \cup B \cup C \cup \bar{A}$, where $\bar{A} = \{\bar{a}_1, \ldots, \bar{a}_P\}$ is a disjoint copy of $A$.

[3] The $N^2$ term corresponds to the time it takes to construct each instance, which consists of $O(N^2)$ edges.

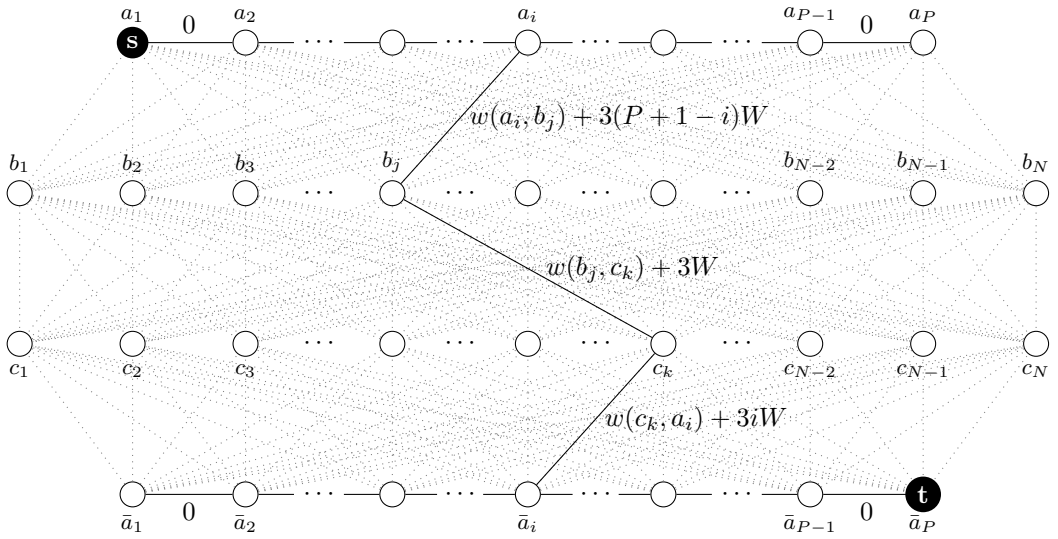



**Figure 2** The graph created in the reduction from Negative Triangle.

- For every $a_i \in A$ and $b_j \in B$, add edge $\{a_i, b_j\}$ with weight $w(a_i, b_j) + 3(P + 1 - i)W$.
- For every $b_j \in B$ and $c_k \in C$, add edge $\{b_j, c_k\}$ with weight $w(b_j, c_k) + 3W$.
- For every $c_k \in C$ and $a_i \in A$, add edge $\{c_k, \bar{a}_i\}$ with weight $w(c_k, a_i) + 3iW$.
- For every $i \in \{1, \ldots, P - 1\}$, add edges $\{a_i, a_{i+1}\}$ and $\{\bar{a}_i, \bar{a}_{i+1}\}$ with weights 0.

Consider a shortest path in this graph from $s \stackrel{\text{def}}{=} a_1$ to $t \stackrel{\text{def}}{=} \bar{a}_P$ using at most $h \stackrel{\text{def}}{=} P + 2$ hops. We claim that its total length is less than $(3P + 2)W$ if and only if there is a negative triangle in the initial graph. Indeed, each triple $(a_i, b_j, c_k) \in A \times B \times C$ corresponds to path $a_1 - a_2 - \cdots - a_i - b_j - c_k - \bar{a}_i - \bar{a}_{i+1} - \cdots - \bar{a}_P$, which uses exactly $P + 2$ hops and has total length

$$w(a_i, b_j) + 3(P + 1 - i)W + w(b_j, c_k) + 3W + w(c_k, a_i) + 3iW$$
$$= \big(w(a_i, b_j) + w(b_j, c_k) + w(c_k, a_i)\big) + (3P + 2)W.$$

Hence, the "if" direction follows. For the "only if" direction, fix an $s$-$t$ path with at most $P + 2$ hops and a total length strictly less than $(3P + 2)W$. The path must be of the form $a_1 - a_2 - \cdots - a_i - b_j - \cdots - c_k - \bar{a}_{i'} - \bar{a}_{i'+1} - \cdots - \bar{a}_P$, where $\{a_i, b_j\}$ is the first edge that leaves $A$ and $\{c_k, \bar{a}_{i'}\}$ is the last edge that enters $\bar{A}$. Every edge has weight at least 0, every edge incident to $b_j$ or $c_k$ has weight at least $-W + 3W = 2W$, and the direct edge between $b_j$ and $c_k$ has weight at most $W + 3W = 2 \cdot 2W$. Thus, the direct edge is the cheapest walk from $b_j$ to $c_k$ both in terms of the length and the number of hops. Consequently, we may assume without loss of generality that our $s$-$t$ path proceeds directly from $b_j$ to $c_k$. This means that the number of hops is $i + 3 + (P - 1 - i') = P + 2 + i - i'$, whereas the total length is

$$w(a_i, b_j) + 3(P + 1 - i)W + w(b_j, c_k) + 3W + w(c_k, a_{i'}) + 3i'W$$
$$= \big(w(a_i, b_j) + w(b_j, c_k) + w(c_k, a_{i'})\big) + (3P + 2)W + 3(i' - i)W.$$

If $i' < i$, then the number of hops is at least $P + 3$, which is larger than assumed. If $i' > i$, then the path length is at least $-3W + (3P + 2)W + 3W \geqslant (3P + 2)W$, a contradiction. Therefore, $i = i'$ holds. Since the path length is less than $(3P + 2)W$, we conclude that $w(a_i, b_j) + w(b_j, c_k) + w(c_k, a_i) < 0$, i.e., $(a_i, b_j, c_k)$ is a negative triangle in the initial graph.

## 3 Hardness under Min-Plus Convolution Hypothesis

**Preliminaries.** In the Max-Plus Convolution Upper Bound problem, we are given three sequences $(a[i])_{i=0}^{n-1}$, $(b[i])_{i=0}^{n-1}$, and $(c[i])_{i=0}^{n-1}$ of $n$ numbers each, and the goal is to decide whether $c[k] \geqslant \max_{i+j=k}(a[i]+b[j])$ holds for all $k$. In other words, we want to find $i, j, k$ such that $i + j = k$ and $c[k] < a[i] + b[j]$. The Max-Plus Convolution Upper Bound and Min-Plus Convolution problems are equivalent under subquadratic reductions [11, Theorem 3.1]; thus, in particular, unless the Min-Plus Convolution Hypothesis fails, there is no $O(n^{2-\varepsilon})$ time algorithm for Max-Plus Convolution Upper Bound, for any $\varepsilon > 0$.

Let us introduce an intermediate problem, which we call Common Max-Plus Convolution Upper Bound: Given $M$ pairs of sequences

$$\big((a_1[i])_{i=0}^{N-1}, (b_1[i])_{i=0}^{N-1}\big), \ldots, \big((a_M[i])_{i=0}^{N-1}, (b_M[i])_{i=0}^{N-1}\big),$$

and one sequence $(c[i])_{i=0}^{N-1}$, decide if there exist $i, j, k, \ell$ such that $i + j = k$ and $c[k] < a_\ell[i] + b_\ell[j]$. We call such $(i, j, k, \ell)$ a *violating quadruple*. First, we show that the naive running time of $O(MN^2)$ is conditionally optimal for this problem.

▶ **Lemma 5.** *Unless the Min-Plus Convolution Hypothesis fails, there is no $O(MN^{2-\varepsilon})$ time algorithm for Common Max-Plus Convolution Upper Bound, for any $\varepsilon > 0$, even when restricted to instances with $M = \Theta(N^\alpha)$ for an arbitrarily chosen constant $\alpha \geqslant 0$.*

**Proof.** The argument is based on a self-reduction of Min-Plus Convolution (see [11, proof of Theorem 5.5]). Let $\beta = \alpha/(1+\alpha) \in [0, 1)$. We start with an instance of Max-Plus Convolution Upper Bound with three sequences $a$, $b$, $c$, each of length $n$. We split $a$ and $b$ into $\Theta(n^\beta)$ blocks of consecutive elements, each block of length $\lceil n^{1-\beta} \rceil$ (the last block can be shorter). For every pair of blocks, one from $a$ and the other from $b$, we want to check if the corresponding fragment of $c$ is an upper bound of their max-plus convolution. Similarly to [11], we add suitable padding so that all three sequences are of the same length. This way, we end up with $\Theta(n^{2\beta})$ smaller instances of Max-Plus Convolution Upper Bound. The key step is to classify these instances according to the third sequence, which results in $\Theta(n^\beta)$ groups of size $\Theta(n^\beta)$ each (the instances in any single group share the same third sequence). Each such group becomes a single instance of Common Max-Plus Convolution Upper Bound, with $M = \Theta(n^\beta)$ and $N = \Theta(n^{1-\beta})$. If each of these instances can be solved in $O(MN^{2-\varepsilon}) = O(n^{\beta+(1-\beta)(2-\varepsilon)})$ time, then the original instance can be solved in $O(n^{2\beta+(1-\beta)(2-\varepsilon)}) = O(n^{2-(1-\beta)\varepsilon})$ time, and the Min-Plus Convolution Hypothesis fails. We conclude the proof by observing that $n^\beta = n^{\alpha/(1+\alpha)} = (n^{1/(1+\alpha)})^\alpha = (n^{1-\beta})^\alpha$, and thus $M = \Theta(N^\alpha)$ holds as desired. ◀

Cygan, Mucha, Węgrzycki, and Włodarczyk [11, proof of Theorem 5.4] showed that, without loss of generality, the input sequences to Max-Plus Convolution Upper Bound are nonnegative and strictly increasing. The same argument applies to Common Max-Plus Convolution Upper Bound. Using the additional structure, we can replace the condition $i+j = k$ with $i+j \leqslant k$. Indeed, suppose we find $(i, j, k, \ell)$ with $i+j \leqslant k$ and $c[k] < a_\ell[i]+b_\ell[j]$; then, by monotonicity, $c[i + j] \leqslant c[k]$, and hence $(i, j, i + j, \ell)$ is a quadruple satisfying the original condition.

▶ **Observation 6.** *The lower bound of Lemma 5 holds even restricted to instances with all the sequences nonnegative and strictly increasing. For such instances, the condition $i + j = k$ can be equivalently replaced by $i + j \leqslant k$.*

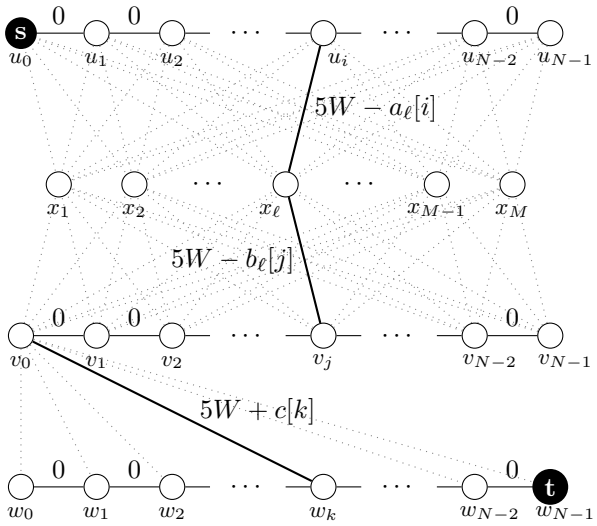



**Figure 3** The graph created in the reduction from Common Max-Plus Convolution Upper Bound.

**Reduction.** We show how to reduce an instance of Common Max-Plus Convolution Upper Bound, with $M$ pairs of length-$N$ sequences, to finding the length of a shortest hop-bounded $s$-$t$ path in an undirected graph with $n = \Theta(N + M)$ nodes, $m = \Theta(NM)$ edges, and the hop bound $h = \Theta(N)$. This will let us conclude that an $O(h^{1-\varepsilon}m)$ time (or $O(hm^{1-\varepsilon})$ time) shortest path algorithm would give an $O(MN^{2-\varepsilon})$ time (respectively, $O(M^{1-\varepsilon}N^{2-\varepsilon})$ time) algorithm for the Common Max-Plus Convolution Upper Bound problem and thus violate the Min-Plus Convolution Hypothesis.

Let $W$ denote the maximum value of any input sequence element. Create an undirected graph (see Figure 3) composed of three paths $u_0 - u_1 - \cdots - u_{N-1}$, $v_0 - v_1 - \cdots - v_{N-1}$, $w_0 - w_1 - \cdots - w_{N-1}$, and an independent set of $M$ nodes $x_1, x_2, \ldots, x_M$. Set the weights of all the path edges to 0. For every $i \in \{0, 1, \ldots, N-1\}$ and $\ell \in \{1, 2, \ldots, M\}$, add an edge between $u_i$ and $x_\ell$ with weight $5W - a_\ell[i]$. Then, for every $j \in \{0, 1, \ldots, N-1\}$ and $\ell \in \{1, 2, \ldots, M\}$, add an edge between $x_\ell$ and $v_j$ with weight $5W - b_\ell[j]$. Finally, for every $k \in \{0, 1, \ldots, N-1\}$, add an edge between $v_0$ and $w_k$ with weight $5W + c[k]$.

Consider a shortest path in this graph from $s \stackrel{\text{def}}{=} u_0$ to $t \stackrel{\text{def}}{=} w_{N-1}$ using at most $h \stackrel{\text{def}}{=} N + 2$ hops. We claim that its total length is less than $15W$ if and only if there is a quadruple $(i, j, k, \ell)$ with $i + j \leqslant k$ and $c[k] < a_\ell[i] + b_\ell[j]$. Indeed, each quadruple $(i, j, k, \ell)$ corresponds to path

$$u_0 - u_1 - \cdots - u_i - x_\ell - v_j - v_{j-1} - \cdots - v_0 - w_k - w_{k+1} - \cdots - w_{N-1}. \qquad (\star)$$

Such a path uses $i + 1 + 1 + j + 1 + (N - 1 - k) = (N + 2) + (i + j - k)$ hops and has total length $15W - a_\ell[i] - b_\ell[j] + c[k]$. Hence, the "if" direction follows. For the "only if" direction, since every non-zero edge weight is at least $4W$, an $s$–$t$ path of total length less than $15W$ can use at most three such edges, and therefore it must be of the form $(\star)$. The hop bound implies $i + j - k \leqslant 0$ and the total length bound implies $c[k] - a_\ell[i] - b_\ell[j] < 0$.

Recall that $n = \Theta(N + M)$, $m = \Theta(NM)$, and $h = \Theta(N)$. For $\eta \in [1/2, 1]$, in order to get hard shortest hop-bounded path instances with density $\Theta(m^\eta)$ and hop bound $h = \Theta(m^\eta)$, we start with the Common Max-Plus Convolution Upper Bound problem restricted to instances with $M = \Theta(N^{(1-\eta)/\eta})$. This implies $h = \Theta(N) = \Theta((NM)^\eta) = \Theta(m^\eta)$. Moreover, due to $\eta \geqslant 1/2$, we have $M \leqslant O(N)$, and thus $n = \Theta(N + M) = \Theta(N) = \Theta(m^\eta)$ holds as desired. This concludes the proof of Theorem 3.

## References


1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *ACM Trans. Algorithms*, 18(1):6:1–6:22, 2022. doi:10.1145/3450524.

2 Anantaram Balakrishnan and Kemal Altinkemer. Using a hop-constrained model to generate alternative communication network design. *INFORMS J. Comput.*, 4(2):192–205, 1992. doi:10.1287/ijoc.4.2.192.

3 Richard Bellman. Notes on the theory of dynamic programming IV - maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, March 1956. doi:10.1002/nav.3800030107.

4 Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958. doi:10.1090/qam/102435.

5 Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 600–611. IEEE, 2022. doi:10.1109/FOCS54457.2022.00063.

6 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/s00453-012-9734-3.

7 Karl Bringmann, Alejandro Cassis, and Nick Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023*. IEEE, 2023. arXiv:2304.05279.

8 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

9 Shigang Chen and Klara Nahrstedt. An overview of quality of service routing for next-generation high-speed networks: problems and solutions. *IEEE Netw.*, 12(6):64–79, 1998. doi:10.1109/65.752646.

10 Gang Cheng and Nirwan Ansari. Finding all hops shortest paths. *IEEE Commun. Lett.*, 8(2):122–124, 2004. doi:10.1109/LCOMM.2004.823365.

11 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

12 Geir Dahl and Luis Eduardo Neves Gouveia. On the directed hop-constrained shortest path problem. *Oper. Res. Lett.*, 32(1):15–22, 2004. doi:10.1016/S0167-6377(03)00026-9.

13 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *12th Annual Symposium on Switching and Automata Theory*, pages 129–131. IEEE Computer Society, 1971. doi:10.1109/SWAT.1971.4.

14 Lester Randolph Ford. Network flow theory. Technical report, RAND Corporation, 1956. URL: https://www.rand.org/pubs/papers/P923.html.

15 Harold N. Gabow. Scaling algorithms for network problems. *J. Comput. Syst. Sci.*, 31(2):148–168, 1985. doi:10.1016/0022-0000(85)90039-X.

16 Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989. doi:10.1137/0218069.

17 Mohsen Ghaffari, Bernhard Haeupler, and Goran Zuzic. Hop-constrained oblivious routing. In *53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021*, pages 1208–1220. ACM, 2021. doi:10.1145/3406325.3451098.

18 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM J. Comput.*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

19 Roch Guérin and Ariel Orda. Computing shortest paths for any number of hops. *IEEE/ACM Trans. Netw.*, 10(5):613–620, 2002. doi:10.1109/TNET.2002.803917.

20 H. C. Joksch. The shortest route problem with constraints. *Journal of Mathematical Analysis and Applications*, 14(2):191–197, 1966. doi:10.1016/0022-247X(66)90020-5.

21 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

22 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53.

23 Jakub Łącki and Yasamin Nazari. Near-optimal decremental hopsets with applications. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 86:1–86:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.86.

24 Alfonso Shimbel. Structure in communication nets. In *Symposium on Information Networks, 1954*, pages 199–203. Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955.

25 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *37th Annual ACM Symposium on Theory of Computing, STOC 2005*, pages 112–119. ACM, 2005. doi:10.1145/1060590.1060607.

26 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *International Congress of Mathematicians, ICM 2018*, pages 3447–3487, 2018. doi:10.1142/9789813272880_0188.

27 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

28 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

# Towards Bypassing Lower Bounds for Graph Shortcuts

**Shimon Kogan**
Weizmann Institute of Science, Rehovot, Israel

**Merav Parter**
Weizmann Institute of Science, Rehovot, Israel

## Abstract

For a given (possibly directed) graph $G$, a hopset (a.k.a. shortcut set) is a (small) set of edges whose addition reduces the graph diameter while preserving desired properties from the given graph $G$, such as, reachability and shortest-path distances. The key objective is in optimizing the tradeoff between the achieved diameter and the size of the shortcut set (possibly also, the distance distortion). Despite the centrality of these objects and their thorough study over the years, there are still significant gaps between the known upper and lower bound results.

A common property shared by almost all known shortcut lower bounds is that they hold for the seemingly simpler task of reducing the diameter of the given graph, $D_G$, by a constant additive term, in fact, even by just one! We denote such restricted structures by $(D_G-1)$-diameter hopsets. In this paper we show that this relaxation can be leveraged to narrow the current gaps, and in certain cases to also bypass the known lower bound results, when restricting to sparse graphs (with $O(n)$ edges):

- **Hopsets for Directed Weighted Sparse Graphs.** For every $n$-vertex directed and weighted sparse graph $G$ with $D_G \geq n^{1/4}$, one can compute an *exact* $(D_G-1)$-diameter *hopset* of linear size. Combining this with known lower bound results for *dense* graphs, we get a separation between dense and sparse graphs, hence *shortcutting sparse graphs is provably easier*. For reachability hopsets, we can provide $(D_G-1)$-diameter hopsets of linear size, for sparse DAGs, already for $D_G \geq n^{1/5}$. This should be compared with the diameter bound of $\widetilde{O}(n^{1/3})$ [Kogan and Parter, SODA 2022], and the lower bound of $D_G = n^{1/6}$ by [Huang and Pettie, SIAM J. Discret. Math. 2018].
- **Additive Hopsets for Undirected and Unweighted Graphs.** We show a construction of $+24$ *additive* $(D_G-1)$-diameter hopsets with linear number of edges for $D_G \geq n^{1/12}$ for sparse graphs. This bypasses the current lower bound of $D_G = n^{1/6}$ obtained for *exact* $(D_G-1)$-diameter hopset by [HP'18]. For general graphs, the bound becomes $D_G \geq n^{1/6}$ which matches the lower bound of exact $(D_G-1)$ hopsets implied by [HP'18]. We also provide new additive $D$-diameter hopsets with linear size, for any given diameter $D$.

Altogether, we show that the current lower bounds can be bypassed by restricting to sparse graphs (with $O(n)$ edges). Moreover, the gaps are narrowed significantly for any graph by allowing for a constant additive stretch.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** Directed Shortcuts, Hopsets, Emulators

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.73

**Funding** This project is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 949083), and by the Israeli Science Foundation (ISF), grant No. 2084/18.

## 1 Introduction

A shortcut set (a.k.a hopset) is a small collection of edges $H$ that when added to a given (possibly directed and weighted) graph $G$ reduces the diameter substantially, while preserving key properties from $G$, such as, reachability, shortest-path distances, etc. Since their


introduction by Ullman and Yannakakis [25] and Thorup [24], shortcut sets have been studied extensively due to their wide-range of applications for parallel, distributed, dynamic and streaming algorithms [16, 12, 10, 15, 9, 11, 3]. Their applicability is also demonstrated by recent algorithmic results, e.g., [15, 9, 11] that use shortcuts as their core component.

We focus on the fundamental notions of *reachability* $D$-diameter hopsets[1] and *exact* $D$-diameter hopsets. These are sets of edges that when added to $G$ reduce the directed diameter to at most $D$ while preserving reachability (resp., exact $D$-hop distances). For a graph on $n$ vertices and $m$ edges, the literature has mainly focused on how small the diameter $D$ can become after adding $\widetilde{O}(n)$ or $\widetilde{O}(m)$ edges to the graph. Recent years have witnessed a significant progress on the combinatorial and algorithmic aspects of these structures. Despite these efforts, there are still major gaps in our understanding. Thorup conjectured [24] that any $n$-vertex digraph with $m$ edges, has a $D$-diameter shortcut of size $\widetilde{O}(m)$ for[2] $D = \widetilde{O}(1)$. This has been shown to hold for a restricted class of graphs, such as planar [24]. Hesse [13] refuted this conjecture for general graphs by presenting a construction of an $n$-vertex digraph with $m = \Theta(n^{19/17})$ edges that requires $\Omega(mn^{1/17})$ shortcut edges to reduce its diameter to below $n^{1/17}$. Huang and Pettie [14] showed a diameter lower bound of $\Omega(n^{1/6})$ for $O(n)$-size shortcuts, and Lu, Vassilevska-Williams, Wein and Xu [21] have recently improved the lower bound for $D$-shortcuts with $O(m)$ edges to $D = \Omega(n^{1/8})$.

Up to very recently, the only known upper bound for reachability hopsets was given by a folklore randomized algorithm, attributed to Ullman and Yannakakis [25], that for every integer $D \geq 1$, provides a shortcut of size $\widetilde{O}((n/D)^2)$. Kogan and Parter [18] have improved the tradeoff to $\widetilde{O}(n^2/D^3 + (n/D)^{3/2})$. Hence, for reachability $D$-diameter hopsets of linear size, we have $D = \widetilde{O}(n^{1/3})$ and $D = \Omega(n^{1/6})$.

The gap becomes even more dramatic when insisting on preserving the *exact* distances. The only known upper bound for exact $D$-diameter hopsets are still given by the folklore algorithm of [25], even for the presumably simpler setting of *unweighted* and *undirected* graphs. Currently, exact $D$-diameter hopsets of linear size require $D = \Omega(n^{1/6})$ for unweighted and undirected graphs, while the upper bound is still $O(\sqrt{n})$. The lower bound for weighted graphs has been recently improved to $D = \Omega(n^{1/3})$ by [17].

In this paper we investigate this curious gap between upper and lower bound results for reachability and exact hopsets. We aim at understanding the limitations of the current lower bound constructions, and study whether they can be bypassed, algorithmically. Our viewpoint differs from the classical algorithmic approach in the sense that we aim to provide algorithms mainly for the purpose of understanding the key barriers for narrowing the observed gaps. In the following, for two given shortcutting tasks $\Pi, \Pi'$, we say that task $\Pi$ is *easier* than task $\Pi'$ if one can provide for task $\Pi$ an improved diameter vs. size tradeoff over the current tradeoff known for task $\Pi'$.

## 1.1 Our Contribution

Our starting observation is that almost all known lower bounds for hopsets, a partial list includes, [13, 14, 21, 17], in-fact hold for the seemingly easier task of reducing the diameter of the given graph by just 1. For example, the lower bound of Huang and Pettie [14] exhibits

[1] These structures are usually referred to as reachability shortcuts, and the notion of hopset is usually used in the context of preserving also the shortest-path distances. For clarity of presentation, we unify the notation and refer to all structures as variants of hopsets.

[2] The $\widetilde{O}(.)$ notation hides poly-log $n$ factors.

an $n$-vertex graph $G$ with diameter $D_G = \Theta(n^{1/6})$ for which any subset of linear number of shortcut edges cannot reduce the diameter to $D_G - 1$. As this property is shared by many of the lower bound results, we raise the following question:

▶ **Question 1.1** (Additive vs. Multiplicative Diameter Reduction). *Is reducing the graph diameter by a constant* ***additive*** *term easier than by a constant* ***multiplicative*** *factor?*

In this context, we say that task $\Pi$ is *easier* than task $\Pi'$, if one can provide a solution for $\Pi'$ which improves the state-of-the-art bounds known for $\Pi$. To answer this question, we focus on the notion of $(D_G - 1)$-diameter hopset: a subset of shortcut edges that reduces the diameter of $G$ from $D_G$ to $D_G - 1$. We provide new algorithmic results for $(D_G - 1)$-diameter hopsets that significantly *narrow* the current gap for these structures.

▶ **Theorem 1.1** (Reachability $(D_G - 1)$-Diameter Hopset). *Every $m$-edge $n$-vertex DAG $G$ admits a reachability $(D_G - 1)$-diameter hopset with $\widetilde{O}(m^{4/3}/D_G^{5/3})$ edges.*

By plugging $m = O(n)$, we get a reachability $(D_G - 1)$-hopset with linear number of edges for every $D_G \geq n^{1/5}$. This suggests that at least algorithmically and for the family of sparse graphs, the task of reducing the diameter additively is easier than reducing the diameter by a constant factor. Currently, for the latter task, linear reachability hopsets exist only for $D_G \geq n^{1/3}$, even for sparse DAGs [18, 19].

Another fundamental feature shared by most of the lower bound results [13, 14, 21, 17] is concerned with the density (i.e. number of edges) of the worst-case graph examples. In particular, all these lower bound graphs have a super-linear number of edges (referred to hereafter as *dense* graphs). E.g., the above-mentioned lower bound graph of [14] contains $\Omega(n^{7/6})$ edges. We therefore raise the following question:

▶ **Question 1.2** (Shortcutting Sparse vs. Dense Graphs). *Is reducing graph diameter easier for* ***sparse*** *graphs compared to* ***dense*** *graphs?*

We answer Question 1.2 in the affirmative by providing a *provable* gap between the sparse and dense settings, in the context of *exact* hopsets. We show:

▶ **Theorem 1.2** (Exact $(D_G - 1)$-Diameter Hopset, Directed and Weighted). *Every $m$-edge $n$-vertex directed (and possibly weighted) graph $G$ admits an exact $(D_G - 1)$-diameter hopset with $\widetilde{O}((m/D_G)^{4/3})$ edges.*

Taking $m = n$ yields $\widetilde{O}(n)$-size exact hopsets for diameter $D_G \geq n^{1/4}$. This should be compared with the lower bound for exact $(D_G - 1)$-diameter hopsets by [4] (one can adapt Theorem 5, therein to this setting) which requires a super-linear number of shortcut edges for dense graphs $G$ with $\Theta(n^{3/2})$ edges and diameter $D_G = n^{1/4}$.

Finally, we address the setting of exact $(D_G - 1)$-diameter hopsets for unweighted and undirected graphs, which currently admits the largest gap between the known upper and lower bound results. Recall that for exact $D$-diameter hopsets of linear size, we have $D = O(\sqrt{n})$. As no progress (at least for upper bounds) has been made for exact hopsets over the years, we consider the question of whether it is possible to narrow this gap by introducing an additive stretch:

▶ **Question 1.3** (Exact vs. Additive Stretch)**.** *Is reducing the graph diameter easier when allowing an* ***additive*** *stretch compared to* ***exact*** *distances?*

To address Question 1.3 we study the notion of additive hopsets, which to the best of our knowledge has not been considered before. For a given graph $G$, an $+\alpha$-additive $D$-diameter hopset is a subset of (weighted) shortcut edges $H$ whose addition guarantees the following: for every $u, v$ pair, the graph $G \cup H$ contains a $D$-hop $u$-$v$ path (that is the path contains at most $D$ edges) of total length[3] $\ell_{u,v}$ where $\text{dist}_G(u,v) \leq \ell_{u,v} \leq \text{dist}_G(u,v) + \alpha$. We show that the introduction of additive stretch can indeed achieve a lot in terms of improving the state-of-the-art bounds when compared to exact hopsets. We have:

▶ **Theorem 1.3** (Additive Hopsets Upper Bounds)**.** *Every $n$-vertex $m$-edge undirected unweighted graph $G$ admits a $+24$-additive $(D_G - 1)$-diameter hopset with $\widetilde{O}(\min\{(n/D_G)^{6/5}, (m/D_G)^{12/11}\})$ edges.*

Theorem 1.3 should be compared with the lower bound of [14]. The latter can be shown to imply that there exists an undirected and unweighted graph $G$ with $\omega(n^{7/6})$ edges for which any *exact* $(D_G - 1)$-diameter hopset requires $\Omega(n)$ edges. For $m = O(n)$, our construction in Theorem 1.3 provides $+24$-additive $(D_G - 1)$-hopset with linear number of edges, for any $D_G \geq n^{1/12}$. That is, by restricting to sparse graphs and introducing a small additive stretch we bypass the lower bound of [14]. Moreover, for any (possibly dense) graph $G$, we provide a linear-size *additive* $(D_G - 1)$-diameter hopset for $D_G \geq n^{1/6}$, hence matching the lower bound of $n^{1/6}$ implied by [14] for *exact* $(D_G - 1)$-diameter hopsets.

We also show a general construction of $\alpha$-additive $D$-hopsets for any given stretch parameter $\alpha$ and diameter bound $D$ (which is possibly much smaller than $D_G$):

▶ **Theorem 1.4.** *Every $n$-vertex undirected unweighted graph $G$ with input parameters $\alpha, D$, admits a $+\alpha$-additive $D$-diameter hopset with $\widetilde{O}((n/(\alpha \cdot D))^2 + (n/\alpha)^{4/3} + n)$ edges. Hence, of linear size for $D, \alpha = \Omega(n^{1/4})$.*

The construction of Theorem 1.4 is based on an interesting connection between hopsets and additive spanners for weighted graphs [1, 7].

**Remark.** In our upper bound results, we focus on $(D_G - 1)$-diameter hopsets since this is a recurrent feature of the state-of-the-art lower bounds. Our algorithms can indeed be easily extended to provide a $-c$ reduction for any given term $c$. In the context of exact hopsets and reachability preservers, the factor $c$ will appear in the final size bound (for constant $c$, the asymptotic size bound remains as is). For additive hopsets the factor $c$ will appear in the final size as well as in the final additive stretch (i.e., the $+24$ term in Theorem 1.3 might become $+24 \cdot c$).

**Recent Breakthrough Lower-Bound Results on Shortcuts and Hopsets.** Following the submission of this paper, Bodwin and Hoppenworth [5] provided new lower bound results for shortcuts and hopsets. In particular, they provide a lower bound of $D = \Omega(n^{1/4})$ for

[3] The length of a path is measured by the sum of its edge weights (note that in an unweighted graph the length is the number of edges in the path).

reachability $D$-diameter hopsets with linear number of edges. Hence, the current gap for these structures is $D = \widetilde{O}(n^{1/3})$ and $D = \Omega(n^{1/6})$. In addition, for exact $D$-hopsets of linear size, they show a lower bound of $D = \Omega(\sqrt{n})$, implying that the folklore algorithm for these structures is indeed tight.

## 1.2 Preliminaries

**Graph Notations.** For an $a$-$b$ dipath $P$ and a $b$-$c$ dipath $P'$ the concatenation of the paths is denoted by $P \circ P'$. Let $|P|$ denote the number of edges (hop) on $P$ and let $\text{len}(P)$ be the *length* of path $P$, measured by the sum of its (possibly) weighted edges. In the context of weighted graphs, we refer to number of edges on a path $P$ as the number of *hops*. For a collection of paths $\mathcal{P}$, let $V(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} V(P)$. For an element set $X$ and $p \in [0,1]$, let $X[p]$ be the set obtained by taking each element of $X$ into $X[p]$ independently with probability $p$. For a subset $V' \subseteq V$, let $G[V']$ be the induced graph on $V'$. For a (possibly directed and weighted) graph $G$ and $u, v \in V(G)$, let $\text{dist}_G(u,v)$ be the shortest-path distance between $u, v$ in $G$. Let $N_{in}(u,G) = \{v \in V(G) \mid (v,u) \in E(G)\}$ and $N_{out}(u,G) = \{v \in V(G) \mid (u,v) \in E(G)\}$. Define $\deg_{in}(u,G) = |N_{in}(u,G)|$ and $\deg_{out}(u,G) = |N_{out}(u,G)|$. For an undirected graph $G$, a vertex $v \in V(G)$ and an integer $k$, let $N_k(v,G) = \{u \in V(G) \mid \text{dist}_G(u,v) \leq k\}$ and let $\deg_k(v,G) = |N_k(v)|$. When $G$ is clear from the context, we may omit it. Also, when $k = 1$, we may write $N(v,G)$, rather than $N_1(v,G)$. For a given path $P \subseteq G$ and $u \in P$, the $k$-hop $P$-neighbor of $u$ is some vertex in $P \cap N_k(u,G)$.

For an $n$-vertex digraph $G$, let $TC(G)$ denote its transitive closure. For $(u,v) \notin TC(G)$, $\text{dist}_G(u,v) = \infty$. A *shortcut* edge is an edge in $TC(G)$. These definitions are naturally extended to *undirected* graphs, by treating all edges as bidirectional, consequently, $TC(G)$ is an $n \times n$ clique for connected undirected graphs. For an integer weighted (possibly directed) graph $G = (V, E, \omega)$ where $\omega : E \to [1, W]$, the weighted transitive closure, denoted as $TC_W(G)$, has the same edge set as $TC(G)$ weighted by the $G$-distances between the edges' endpoints. For any vertices $u, v \in V$ and integer $\beta \leq n$, define $\text{dist}^{(\beta)}_G(u,v)$ to be the minimum length $u$-$v$ path with at most $\beta$ edges (hops). If there is no such path, then define $\text{dist}^{(\beta)}_G(u,v) = \infty$. Throughout, we define the *diameter* of the graph, $D_G$, by the smallest value $\beta$ satisfying that $\text{dist}^{(\beta)}_G(u,v) = \text{dist}_G(u,v)$ for every $u, v \in V$. In the context of weighted graphs $G$, $D_G$ as defined above is usually refereed to as the *hopbound*. For ease of readability, we slightly revise the notion of diameter in a way the captures weighted and unweighted hopsets as well as reachability shortcuts.

**Reachability, Exact and Additive Hopsets.** For a directed graph $G$, a reachability $d$-diameter hopset $H$ is a subset of edges from the transitive closure $TC(G)$ such that $D_{G \cup H} \leq d$. For a (possibly directed and weighted) graph $G$, an $\alpha$-additive $d$-diameter hopset $H$ is a subset of weighted edges in $TC_W(G)$ satisfying that for every $u, v \in V(G)$:

$$\text{dist}_G(u,v) \leq \text{dist}^{(d)}_{G \cup H}(u,v) \leq \text{dist}_G(u,v) + \alpha \ . \tag{1.1}$$

An *exact* $d$-diameter hopset satisfies Eq. (1.1) for $\alpha = 0$.

In this paper, we study the notion of $(D_G - 1)$-diameter hopsets which given a graph $G$ provide the desired hopbound guarantees for $d = D_G - 1$. Our algorithms can easily provide any constant additive reduction, i.e., a $(D_G - c)$-diameter hopsets, for any desired constant $c$. We use the notion of $(D_G - 1)$ as a reference to the special property of the current lower bounds. That is, $(D_G - 1)$-hopsets are precisely the structures for which we currently have lower bounds for. We note that our bounds also hold in the following alternative formulation:

Given an input parameter $D$, then the output hopset guarantees a hopbound reduction by 1 (or by a constant term) for any $u$-$v$ shortest path with hopbound at least $D$. That is, the reduction can be provided for all paths of length $\geq D$ and not only for $D = D_G$.

**Shortcutting Tools.** We use the basic shortcutting algorithm for dipaths from [23].

▶ **Lemma 1.5** (Restatement of Lemma 1.1 in [23]). *Any dipath $P$ admits a reachability 2-diameter hopset $H$ with $\widetilde{O}(|P|)$ edges.*

▶ **Lemma 1.6** ([25]). *For every $n$-vertex (possibly directed and weighted) graph $G$ and integer $D \leq n$, there is an algorithm* ExactHopset *that computes an exact $D$-diameter hopset $H$ with $\widetilde{O}((n/D)^2)$ edges.*

Our constructions employ a useful variant of reachability hopsets, in which it is required to provide the desired hopbound w.r.t to a given subset of vertices [19].

▶ **Definition 1.1** (Subset Reachability Hopset [19]). Given a graph $G = (V, E)$, a subset $S \subseteq V$ and an integer $D$, a set of edges $H \subseteq TC(G)$ is an $(S, D)$-reachability-hopset, if for every $u, v \in V$ such that $(u, v) \in TC(G)$, there is a $u$-$v$ path $P_{u,v}$ (not necessarily a shortest path) in $G \cup H$ with at most $D$ vertices from $S$.

For completeness we provide a complete proof for the following which also provides an additional property compared to the construction of [19]. See Appendix A for the proof.

▶ **Lemma 1.7.** *For every $n$-vertex DAG $G$, $S \subseteq V(G)$ and input parameter $D$, one can compute an $(S, D)$-reachability-hopset $H \subseteq TC(G)$ of size $\widetilde{O}(|S| + (|S|^2/D^3))$. In addition, the hopset $H$ satisfies the following for every $u, v \in V$: If there is a shortest path $P_{u,v} \subseteq G$ which contains $k$ vertices in $V \setminus S$, then in $G \cup H$, there is a $u$-$v$ path (which is not necessarily the shortest path) that contains at most $D$ vertices from $S$ and at most $k$ vertices from $V \setminus S$.*

**Spanners and Emulators.** Graph spanners introduced by Peleg and Schäffer [22] are sparse *subgraphs* that preserve shortest path distances up to a small stretch. In contrast to hopsets, these structures are defined only for *undirected* graphs, as no sparsificiation is possible in the worst-case for directed $n$-vertex graphs with $\Omega(n^2)$ edges.

▶ **Definition 1.2** ($(\alpha, \beta)$-Spanner). Given an undirected graph $G = (V, E)$, a subgraph $G^* \subseteq G$ is called an $(\alpha, \beta)$-*spanner* if $\text{dist}_{G^*}(u, v) \leq \alpha \cdot \text{dist}_G(u, v) + \beta$ for every $u, v \in V$.

An $(\alpha, \beta)$-*emulator* $E^*$ is a weighted set of edges in $V \times V$ (i.e., not necessarily a subgraph of $G$) that provides the same stretch guarantees as $(\alpha, \beta)$-spanners, where $\text{dist}_G(u, v) \leq \text{dist}_{E^*}(u, v) \leq \alpha \cdot \text{dist}_G(u, v) + \beta$ for every $u, v \in V$.

Our constructions also use recent algorithms for computing additive spanners for weighted graphs [1, 8]. For a weighted graph $G = (V, E, \omega)$ where $\omega : E \to \{1, \ldots, W\}$, these spanners provide a $+\beta \cdot W$ stretch guarantees. We use the following theorem for weighted additive spanners by [2] (recently improved by [7]).

▶ **Theorem 1.8** (Theorem 3 in [2]). *For any $n$-vertex weighted graph $G = (V, E, \omega)$ with maximum edge weight $W$, there is an algorithm* WeightedSpanner *that computes a $+8W$ additive spanner $H \subseteq G$ with $\widetilde{O}(n^{4/3})$ edges.*

## 2 Directed Shortcuts

In this section we observe that the existing lower bounds for directed hopsets hold for the relaxed task of $(D_G - 1)$-diameter hopsets. We then show that these lower bounds can be bypassed for sparse graphs. Our upper bounds yield a separation between sparse and dense graphs, implying that shortcutting *sparse* graphs might be *simpler* in terms of providing an improved diameter vs. size tradeoffs.

### 2.1 Exact $(D_G - 1)$-Hopsets

**Known Lower Bound.** The following lower-bound follows by plugging $\ell = n^{1/3}$ in Theorem 5 of [4]. See also Table 3 in [6]. We observe that this lower bound argument, as all prior lower bounds, holds even when it is required to reduce the diameter of the given graph by 1.

▶ **Theorem 2.1** (Exact Hopsets, Directed, Follows by Theorem 5 of [4]). *There exist an $n$-vertex (dense) directed graphs with $\Theta(n^{3/2})$ edges for which any exact $(D_G - 1)$-hopset must have $\Omega(n^{5/4})$ shortcut edges provided that $D_G \leq n^{1/4}$.*

**New Upper Bound (Proof of Theorem 1.2).** We show that the lower bound of Theorem 2.1 can be bypassed for sparse graphs while preserving the exact distances in a weighted digraph. Note that for $m = n$, Theorem 1.2 yields $O(n)$-size exact hopsets for diameter $D \geq n^{1/4}$ while the lower bound of Theorem 2.1 for exact $(D_G - 1)$-diameter requires a super-linear size for $D = n^{1/4}$.

**The Construction.** Set an integer threshold $k = \lceil (m/D_G)^{1/3} \rceil$ and define a vertex $u$ to be *low-deg* if $\deg_{in}(u) \leq k$ and $\deg_{out}(u) \leq k$. Otherwise, the vertex is *high-deg*. By a simple counting, there are $O(m/k)$ high-deg vertices in the given $m$-edge graph.

Let $L = V[p]$ for $p = \Theta(\log n/D_G)$ be a random sample of vertices, and let $L_h, L_\ell$ be the sets of high-deg (resp., low-deg) vertices in $L$ (hence, $L_h \cup L_\ell = L$). The algorithm adds *two* subsets of shortcut edges $H_h, H_\ell$ which handle the high-deg and low-deg, respectively:

- $H_h = (L_h \times L_h) \cap TC(G)$ .
- For every $u \in L_\ell$, $H_\ell(u) = \{(x, y) \mid x \in N_{in}(u), y \in N_{out}(u)\}$.
- $H_\ell = \bigcup_{u \in V} H_\ell(u)$.

All added shortcut edges in $H_h, H_\ell$ are weighted by their shortest-path distances in $G$. The output hopset is given by $H = H_h \cup H_\ell$. This completes the description of the construction.

**Proof of Theorem 1.2.**

**Size.** Since there are $O(m/k)$ high-deg vertices, w.h.p., $|L_h| = O(m \log n/(kD_G))$. Hence, $|H_h| = \widetilde{O}((m/(kD_G))^2)$. We next bound the size of $H_\ell$. Let $V_\ell$ be the set of all low-deg vertices in $G$. Then, $\sum_{u \in V_\ell} \deg_{in}(u) \cdot \deg_{out}(u) = O(km)$. Since $L_\ell$ is obtained by sampling each low-deg vertex with probability of $\Theta(\log n/D_G)$, we get that w.h.p., $|H_\ell| = \widetilde{O}(km/D_G)$. The size argument holds by setting $k = \lceil (m/D_G)^{1/3} \rceil$.

**Diameter and Distances.** Consider some $u$-$v$ shortest-path $P \subseteq G$ with $D_G$ edges. First assume that $P \cap L_\ell \neq \emptyset$. I.e., that $P$ contains at least one sampled low-deg vertex, say $z$. Let $z', z''$ be the vertex preceding (resp., subsequent to) $z$ on the path $P$. Since $z' \in N_{in}(z)$ and $z'' \in N_{out}(z)$, the shortcut edge $(z', z'')$ is in $H_\ell$. Consequently, the path $P$ can be shortcut into a path $P'$ obtained by replacing the 2-hop segment $P[z', z''] = (z', z) \circ (z, z'')$ by a weighted edge $(z', z'')$. Note that $\text{len}(P') = \text{len}(P)$ and that $|P'| = |P| - 1$.

From now on assume that $P$ has no sampled low-deg vertex. W.h.p., it then holds that $P$ contains at least two high-deg sampled vertices, i.e., $|P \cap L_h| \geq 2$. In addition, we can assume that those two sampled vertices, $x, y$ are at hop-distance at least $D/3$ from each other. Since the weighted edge $(x, y) \in H_h$, the $\Omega(D)$-hop segment $P[x, y]$ can be replaced by the single edge $(x, y)$. The distances are clearly preserved. Note that in this case, the hopbound can be reduced by an even a constant factor, and hence the additive reduction is bottle-necked by the low-deg vertices. ◀

## 2.2 Reachability $(D_G - 1)$-Hopsets

**Known Lower Bound.** We start by making the immediate observation that the well-known lower-bound by Huang and Pettie [14] also holds for $(D_G - 1)$-hopsets.

▶ **Theorem 2.2** (Reachability Hopsets, Slight Restatement of [14]). *There exists an $n$-vertex directed acyclic graph with $\Omega(n^{7/6})$ edges for which any reachability $(D_G - 1)$-hopset must have $\Omega(n)$ shortcut edges provided that $D_G \leq n^{1/6}$.*

**Reachability $(D_G - 1)$-Diameter Hopsets (Proof of Theorem 1.1).** When settling for reachability, rather than exact distances, one can provide improved bounds. In particular, Theorem 1.1 claims that sparse DAGs admit a reachability $(D_G - 1)$-diameter hopset of linear size for $D_G \geq n^{1/5}$ (compared to $D_G \geq n^{1/4}$ when preserving the exact distances).

Throughout, recall that $G$ is a DAG[4]. The algorithm for reachability $(D_G - 1)$-hopsets is similar to that of Theorem 1.2. The main distinction is that instead of connecting each pair of sampled high-deg vertices in the hopset, we add the subset reachability hopset of Lemma 1.7 w.r.t. to the set of high-degree vertices. In our context, this adds $|L_h|^2/D_G^3$ edges, rather than $|L_h|^2$ edges, where $L_h$ is the set of sampled high-degree vertices in the construction of Theorem 1.2.

**Proof of Theorem 1.1.** Set $k = \lceil m^{4/3}/D^{5/3} \rceil$. The definition of $H_\ell$ is the same, hence $|H_\ell| = \widetilde{O}(mk/D) = \widetilde{O}(m^{4/3}/D_G^{5/3})$. The subset $H_h$ is defined by applying the subset reachability hopset of Lemma 1.7 with $S = L_h$ and $D = D_G/2$. Since $|S| = \widetilde{O}(m/(kD))$, the size of $H_h$ can be bounded by $\widetilde{O}(|L_h| + (|L_h|^2/D_G^3)) = \widetilde{O}(m^{4/3}/D_G^{5/3})$.

It remains to consider the diameter argument. By the proof of Theorem 1.2 it is sufficient to consider a $u$-$v$ shortest path $P$ of hopbound $D_G$ that has at least $D_G/3$ high-deg vertices and at most $D_G/3$ low-deg vertices. By Lemma 1.7, we have that $H_h$ contains a $u$-$v$ path with at most $D_G/3$ low-deg vertices and at most $D_G/3$ high-deg vertices. Note that in this case, we get a constant reduction in the diameter. The theorem follows. ◀

# 3 Additive Shortcuts for Undirected Unweighted Graphs

We next consider the gap obtained for exact hopsets in undirected and unweighted graphs. Our goal is to narrow this gap by (i) restricting attention to $(D_G - 1)$-hopsets (for which the known lower bound results hold), and (ii) allow a constant additive stretch in the distances. We will show that for sparse graphs, one can even bypass the current lower bound obtained for exact hopsets, as the latter is based on a dense graph example.

[4] Note that the general DAG reduction introduces a factor of 2 in the diameter, and hence we cannot employ it. All lower-bound graphs for this problem are DAGs as well.

**Known Lower Bounds for Exact $(D_G-1)$-Hopsets.** We start by observing that the lower bounds for reachability hopsets by Huang and Pettie [14] also hold for exact hopsets for undirected and unweighted graphs. See Appendix A for a proof.

▶ **Theorem 3.1** (Lower Bound for Exact Hopsets, Undirected Unweighted, Immediate by [14]). *There exist $n$-vertex undirected and unweighted graphs $G$ with $m=\Omega(n^{7/6})$ edges and $D_G=\Theta(n^{1/6})$, such that any exact $(D_G-1)$-diameter hopset for $G$ has $\Omega(n)$ edges.*

We next show that the lower bound of Theorem 3.1 can be bypassed for sparse graphs when allowing additive stretch. In addition, for general graphs (possibly dense) we match the bound of $n^{1/6}$ obtained for exact hopsets.

**New Additive $(D_G-1)$-Hopsets (Proof of Theorem 1.3).** We start by presenting an algorithm that achieves a size bound of $\widetilde{O}((n/D_G)^{6/5})$ (hence, linear-size for $D_G\geq n^{1/6}$) and then explain how to modify the construction to yield the bound of $\widetilde{O}((m/D_G)^{12/11})$ (which provides the improved results for sparse graphs). For the sake of this extension, we show the following slightly stronger statement:

▶ **Lemma 3.2.** *Every $n$-vertex $m$-edge undirected unweighted graph $G$ and any input parameter $D$, one can compute a hopset $H$ with $\widetilde{O}((n/D)^{6/5})$ edges with the following guarantee: For any $u,v$ pair at distance at least $D$ in $G$, it holds that* $\mathrm{dist}_G(u,v)\leq \mathrm{dist}^{(D-1)}_{G\cup H}(u,v)\leq \mathrm{dist}_G(u,v)+24$.

Note that Lemma 3.2 in particular implies a $+24$-additive $(D_G-1)$-diameter hopsets with $\widetilde{O}((n/D)^{6/5})$ edges. The lemma is stronger in the sense that for any given $D$ (where possibly $D<D_G$), the $(-1)$ reduction in the hopbound holds for any shortest path of length at least $D$. In contrast, $(D_G-1)$-hopsets provides the $(-1)$ reduction only for shortest-paths of length *exactly* $D_G$.

**The Construction.** Let $\deg_2(u)$ be the number of 2-hop neighbors of $u$ in the given graph $G$. Set $k=(n/D)^{1/5}$ as a parameter that serves as our 2-degree threshold, as follows. A vertex $u$ is *low-deg* if $\deg_2(u)\leq k$ and it is *high-deg* otherwise. The algorithm has two phases. The first phase handles the low-deg vertices by adding $\widetilde{O}(nk/D)$ shortcut edges. At the end of that phase, the algorithm outputs also a subgraph $G'$ in which each vertex is high-deg. The second phase handles these high-deg vertices by adding an additional subset of $\widetilde{O}(nk/D+(n/(Dk^2))^2)$ shortcut edges. The size bound of $(n/D)^{6/5}$ is obtained by balancing these two size terms, which is achieved for $k=(n/D)^{1/5}$. Throughout, all added shortcut edges are weighted by the corresponding shortest-path distances between the edge endpoints.

**Step (1): Handling Low-Degree Vertices.** The algorithm iterates over the low-deg vertices in the graph, as long as such exists. Initially, set $G_0=G$ and $H_0=\emptyset$. In every iteration $i\geq 1$, it gets as input a subgraph $G_{i-1}$ and $H_{i-1}$ and considers an arbitrary low-deg vertex $u$. (If no such exists, then $G'=G_{i-1}$, $H'=H_{i-1}$ and the step terminates). First, the algorithm removes $u$ by letting $G_i=G_{i-1}\setminus\{u\}$. Then, with probability of $p=\Theta(\log n/D)$, the algorithm connects $u$ to each of its (current) 2-hop neighbors in $G_{i-1}$ by letting $H_i=H_{i-1}\cup\{(u,v)\ \mid\ v\in N_2(u,G_{i-1})\}$. This completes the description of the $i^{th}$ iteration. Denoting the number of iterations by $\ell$, then the output of the step is given by $G'=G_\ell$ and $H'=H_\ell$.

**Step (2): Handling the Remaining High-Degree Vertices.** We now restrict attention only to the graph $G'$. Letting $V' = V(G')$, the algorithm computes *three* random samples of $V'$-vertices: $S = V'[p]$, $Q = V'[q]$ and $R = V'[r]$ where $p = \Theta(\log n/D)$, $q = \Theta(\log n/k)$ and $r = \Theta(\log n/(Dk^2))$. Also, initially set $H'' = (R \times R)$, and add to $H''$ a subset of shortcut edges by applying the following shortcutting procedure for every vertex $u \in S$:

- Build a depth-6 BFS tree $T_6(u, G') \subseteq G'$ rooted at $u$ (i.e., a BFS which spans only $N_6(u, G')$).
- If $|T_6(u, G')| = O(k^2)$, add the edges $E'(u) = \{(u, v) \mid v \in T_6(u, G') \cap Q\}$ to $H''$.

The output hopset is given by $H' \cup H''$. We next turn to analyze the construction and prove Lemma 3.2.

**Size.** In the first step, for every low-deg vertex $u$, with probability of $p = \Theta(\log n/D)$, the algorithm adds $\deg_2(u) \leq k$ shortcut edges. Hence this adds $|H'| = \widetilde{O}(nk/D)$ edges, w.h.p. Consider the second step. W.h.p., $|S| = O(n \log n/D)$ and $|R| = O(n \log n/(Dk^2))$. For every $u \in S$ with $|T_6(u, G')| = O(k^2)$, we have that w.h.p. $|T_6(u, G') \cap Q| \leq k \log n$, and therefore, we have $O(|S| \cdot k \log n)$ edges, due to this step. Overall, we added to $H''$ a total of $O(|R|^2 + |S| \cdot k \log n)$ edges. The size bound follows by plugging $k = (n/D)^{1/5}$.

**Diameter and Stretch Analysis.** Throughout, we override notation and redefine a vertex $v$ to be low-deg if $v \notin V(G')$. That is, a vertex is low-deg if there exists an iteration $i$ in the Step (1) in which $N_2(v, G_{i-1}) \leq k$. A vertex $v$ is then high-deg if $v \in V(G')$ (i.e., it is a high-deg in each subgraph $G_{i-1}$ considered in each iteration $i$ of Step (1)). We also need the following classification of the high-deg vertices. A vertex $u \in V(G')$ is *large* if its 6-depth BFS tree has size $|T_6(u, G')| = \omega(k^2)$, and it is *small* otherwise. Let $V_\ell = V \setminus V(G')$ be the low-deg vertices, $V_h^{small}$ (resp., $V_h^{large}$) denote the small (resp., large) high-deg vertices. It then holds that $V_\ell \cup V_h^{small} \cup V_h^{large} = V(G)$.

Consider now a $u$-$v$ shortest-path $P$ with $D$ edges (hops). The analysis breaks into three cases depending on the number of vertices in $V(P)$ that belong to each of the three subsets of vertices $V_\ell, V_h^{small}$ and $V_h^{large}$.

**Case 1: $|P \cap V_\ell| = \Omega(D)$.** For every low-deg vertex $z$, let $i_z$ be the iteration in which $z$ is removed in Step (1). I.e., $z$ is the (unique) selected low-deg vertex in $G_{i_z-1}$. We then say that the low-deg vertex $z \in P \cap V_\ell$ is *bad* if both of its 2-hop $P$-neighbors $z_1, z_2$ are not in $G_{i_z-1}$. This can happen if these two 2-hop neighbors were removed in prior iterations. Otherwise, the vertex is *good*.

▶ **Lemma 3.3.** *The path $P$ can contain at most* two *consecutive bad vertices.*

**Proof.** The claim follows by noting that a 2-hop $P$-neighbor of a bad vertex $x \in P$ must be good. To see this, let $z$ be a 2-hop $P$-neighbor of $x$. Since $x$ is bad, it implies that $i_x \geq i_z + 1$, i.e., $z$ is removed *before* $x$ in Step (1). This implies that $x \in G_{i_z-1}$ and hence $z$ must be good. ◀

By Lemma 3.3, we get that in this case $P$ contains $\Omega(D)$ good vertices. Since the algorithm add shortcut edges to each good vertex w.p. $p = \Theta(\log n/D)$, we get that w.h.p., the coin flip is successful for at least one good vertex, say $x$, on $P$. By the definition of the good vertex, at least one of its 2-hop $P$-neighbors, say $x'$, is in $G_{i_x-1}$ and therefore the shortcut edge $(x', x)$ is in $H'$. Let $P'$ be the path obtained by replacing the 2-hop segment $P[x, x']$ with an edge $(x, x')$. Then, $\text{len}(P) = \text{len}(P')$ but $|P'| \leq |P| - 1$, as required.

**Case 2.1: $|P \cap V_h^{large}| = \Omega(D)$.** Note that for any for any two vertices $u', u''$ on $P$ at distance at least 13 from each other, it holds that their 6-hop neighborhoods are disjoint. Also note that since $N_6(u', G') \subseteq N_6(u', G)$, we have that $|N_6(u', G)| = \omega(k^2)$ for every $u' \in V_h^{large}$. As $P$ contains $\Omega(D)$ vertices from $V_h^{large}$, we get that the size of the 6-hop neighborhood of the path $P$ is $\Omega(Dk^2)$. Since we sample each vertex in $V$ into $R$ independently with probability of $r = \Theta(\log n/(Dk^2))$, we get that w.h.p., the following holds: There are two vertices $u', u'' \in S \cap P \cap V_h^{large}$ at distance $\Omega(D)$ from each other such that there exists $w \in N_6(u', G) \cap R$ and $w' \in N_6(u'', G) \cap R$. Since the algorithm adds to $H''$ the shortcut edge $(w, w')$, the hopbound between $u$ and $u'$ is reduced from $\Omega(D)$ to at most 13. By the triangle inequality, this introduces an additive stretch of at most $+24$. See Fig. 1 (Left) for an illustration.

**Case 2.2: $|P \cap V_h^{small}| = \Omega(D)$.** Since $P$ has $\Omega(D)$ vertices from $V_h^{small}$, we can also assume the following. The path $P$ contains $\ell = \Omega(D)$ segments $P_1, \ldots, P_\ell$ such that: (i) each $P_i \subseteq G'$, (ii) $|P_i| = 40$ and (iii) the internal 20-length segment of $P_i$ contains a vertex in $V_h^{small}$. We then get that w.h.p. there exists a vertex $u' \in S \cap P \cap V_h^{small}$ that belongs to the internal 20-length segment of some $P_i \subseteq G'$. Let $w$ be a vertex at distance 4 from $u'$ on $P_i$. Since each vertex in $G'$ has 2-deg at least $k$, we get (i) $|N_2(w, G')| \geq k$ and (ii) $N_2(w, G') \subset T_6(u', G')$. Therefore, w.h.p., it holds that there exists some $z \in N_2(w, G') \cap Q$ (where $Q = V'[\Theta(\log n/k)]$) and consequently, $(u', z) \in E'(u)$. The 4-hop path segment $P[u', w]$ can then be replaced by a 3-hop segment $P' = (u', z) \circ (z, z') \circ (z', w)$ for some $z' \in N(w, G') \cap N(z, G')$. It is easy to see that the additive stretch is at most $+4$, as we replace a 4-hop segment by a 3-hop segment of length at most 8. See Fig. 1 (Right) for an illustra

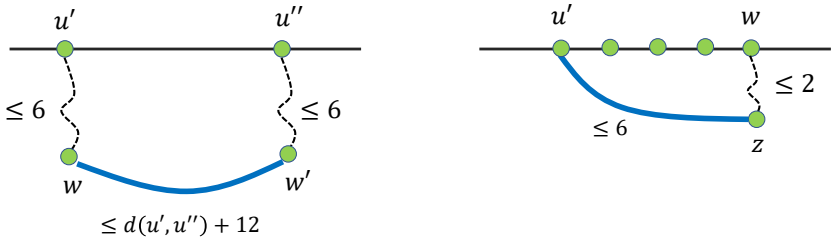



**Figure 1** An illustration for stretch and diameter argument of Theorem 1.3. Left: The path $P$ contains $\Omega(D)$ vertices which are high-deg and an in addition with large 6-depth BFS trees. The shortcut edge is shown in blue. Right: The path $P$ has $\Omega(D)$ vertices which are high-deg and with small 6-depth BFS trees.

We are now ready to complete the proof of Theorem 1.3.

**Theorem 1.3.** It remains to modify the construction to obtain a size bound of $\widetilde{O}((m/D_G)^{12/11})$ edges. We start with a preliminary sparsification step that handles vertices with 1-degree (that is simply the degree) at most $k' = (m/D_G)^{1/11}$. Let $V_{\ell,1}$ be the subset of all low-degree vertices (i.e., with 1-deg at most $k'$) and let $V_{h,1} = V \setminus V_{\ell,1}$. Let $H_1$ be a subset of shortcut edges that handles the low-degree vertices, as follows. Let $V'_{\ell,1} = V_{\ell,1}[p]$ for $p = \Theta(\log n/D_G)$. Then,

$$H_1 = \bigcup_{v \in V'_{\ell,1}} \{(x, y) \mid x, y \in N(v) \text{ and } x \neq y\} .$$

Next, we apply the construction of Lemma 3.2 on the graph $G_h = G[V_{h,1}]$ with $D = D_G$, which outputs a $+24$-additive $(D_G - 1)$-diameter hopset $H_2$ on the graph $G_h$. Observe that the diameter of $G_h$ might be considerably larger than $D$. This motivates the more dedicated guarantees of Lemma 3.2. The output hopset is given by $H_1 \cup H_2$.

**Size.** By the Chernoff bound, w.h.p., $|H_1| = \widetilde{O}(mk'/D_G)$. In addition, $|V_{h,1}| = O(m/k')$ by a simple counting argument. Letting $n' = |V_{h,1}| = O(m/k')$, by Lemma 3.2, $|H_2| = \widetilde{O}((n'/D_G)^{6/5})$. This size bound follows by plugging $k' = (m/D_G)^{1/11}$.

**Diameter and Stretch.** Consider a $u$-$v$ shortest-path $P$ with $D_G$ edges. If $P$ contains $\Omega(D_G)$ vertices of degree at most $k'$, then w.h.p., $P \cap V'_{\ell,1} \neq \emptyset$, and $P$ is shortcut by one hop, and the distances are preserved. It remains therefore to consider the case where $P$ contains $\ell = \Omega(D_G)$ segments $P_1, \ldots, P_\ell$, each of length 100, that are fully contained in the graph $G_h = G[V_h]$. For every $i \in \{1, \ldots, \ell\}$, letting $P_i = [u_{i1}, \ldots, u_{i100}]$ then define an internal segment $P'_i = [u_{i50}, \ldots, u_{i80}] \subset P_i$.

We next show that this suffices to recover the same argument as obtained in Lemma 3.2. Partition the vertices on $P \cap V_{h,1}$ into three subsets $V_\ell, V_h^{small}, V_h^{large}$ as in the argument of Lemma 3.2. We then consider the same cases as in Lemma 3.2 with minor modifications.

**Case 1: $V_\ell$ intersects with $\Omega(D_G)$ distinct segments of $P'_1, \ldots, P'_\ell$.** By Lemma 3.3, we get that w.h.p. the algorithm adds a shortcut edge between some vertex $V_\ell \cap P$ to its 2-hop neighbor on the path. Hence, the diameter (hopbound) is reduced by 1, and the distances are preserved.

**Case 2.1: $V_h^{large}$ intersects with $\Omega(D_G)$ distinct segments of $P'_1, \ldots, P'_\ell$.** Note that any for any two vertices $u, v$ on $P$ at distance at least 13 from each other, it holds that their 6-hop neighborhoods are disjoint. This implies that the 6-hop neighborhood of the path $P$ is $\Omega(Dk^2)$. By the exact same argument as obtained for Case 2.1 in Lemma 3.2, we get that there is at least one segment $P_j$ whose hopbound is reduced while introducing an additive stretch of at most $+24$. This holds as the argument in Lemma 3.2 is local in the sense that it shortcuts segments of constant length provided that there are sampled $u', u'' \in V_h^{large}$ that are sufficiently apart on $P$ and that each has a 4-hop $P$-neighbor on the segment.

**Case 2.2: $V_h^{small}$ intersects with $\Omega(D_G)$ distinct segments of $P'_1, \ldots, P'_\ell$.** The claim follows by noting that the argument for Case 2.2 of Lemma 3.2 shows that any segment $P_j$ gets shortcut (and while introducing an additive stretch of $+4$) with probability of $\Theta(\log n/D_G)$. The probabilities are independent for vertex-disjoint segments. Note that this holds since it is sufficient for the segment $P_j$ to include the 6-hop neighborhood of the vertex. Since $V_h^{small}$ intersects with $\Omega(D_G)$ disjoint segments, at least one of them is successful, w.h.p., and provides the desired $(-1)$ reduction in the hopbound (while introducing an additive stretch of $+4$). ◀

**New Additive $D$-Diameter Hopsets (for any $D$).** We now turn to proving Theorem 1.4. For simplicity we show a construction of $O(D)$-diameter hopsets with additive stretch $O(\alpha)$, but these constant factors can be easily omitted. The algorithm has two main steps. The first step computes a graph $G'$ on $O(n \log n/\alpha)$ vertices at the cost of introducing an additive stretch of $O(\alpha)$. The second step computes an exact $D$-diameter hopset on $G'$.

Specifically, the algorithm starts by computing a weighted *net* graph $G' = \mathsf{Net}(G, p)$ for the given (unweighted) graph $G$ where $p = \Theta(\log n/\alpha)$. The algorithm $\mathsf{Net}(G, p)$ outputs a graph $G' = (V', E', \omega')$ defined as follows. Let $V' = V[p]$ be a random sample of $V$, obtained by sampling each $v \in V$ independently with probability of $p$. Let $E' = \{(u, v) \in V' \times V' \mid \text{dist}_G(u, v) \leq \Theta(\log n/p) \cdot W\}$ and $\omega'((u, v)) = \text{dist}_G(u, v)$ for every $(u, v) \in E'$. We use the following observation in our constructions:

▶ **Observation 3.1** (Observation 3.4 in [20]). Let $G' = (V', E', \omega')$ be the output net graph of Alg. $\mathsf{Net}(G, p)$ where $G = (V, E, \omega)$ is an $n$-vertex graph with maximum edge weight $W$. Then w.h.p., the following holds: (i) $|V'| = O(np \log n)$, (ii) for every $u, v \in V'$, $\text{dist}_{G'}(u, v) = \text{dist}_G(u, v)$, and (iii) the maximum edge weight of $G'$ is bounded by $W' = \Theta(W \log n/p)$.

Denote $S_1 = V(G')$, hence $S_1 = V[p]$. By Obs. 3.1(iii), the maximum edge weight of $G'$ is $W = O(\alpha)$. Let $S_2 = S_1[q]$ for $q = \Theta(\log n/D)$. For a vertex $u$ and a set $S_1$, let $\mathit{Closest}(u, S_1)$ be the closest vertex to $u$ in $S_1$, breaking ties arbitrarily. The output hopset $H$ is the union of $H_0 \cup H_1 \cup H_2$ of weighted edges, where:

- $H_0 \leftarrow \{(u, \mathit{Closest}(u, S_1)) \mid u \in V\}$.
- $H_1 \leftarrow \mathsf{WeightedSpanner}(G')$ of Theorem 1.8.
- $H_2 \leftarrow \mathsf{ExactHopset}(H_1, D)$ of Lemma 1.6.

This completes the description of the hopset.

**Proof of Theorem 1.4.** Clearly, $|H_0| \leq n$. By the Chernoff bound, w.h.p., $|S_1| = O(n \log^2 n/\alpha)$, and therefore by Theorem 1.8, $|H_1| = \widetilde{O}((n/\alpha)^{4/3})$. The size of $H_2$ can bounded by $|S_2|^2 = \widetilde{O}((n/(\alpha \cdot D))^2)$, w.h.p. We next turn to consider the additive stretch and the hopbound.

Consider a $u, v$ pair and let $u' = \mathit{Closest}(u, S_1)$ and $v' = \mathit{Closest}(v, S_1)$. W.h.p., it holds that $\text{dist}_G(u, u'), \text{dist}_G(v, v') = O(\alpha)$. Since, $u', v' \in S_1$, by Obs. 3.1 we have that $\text{dist}_{G'}(u', v') = \text{dist}_G(u, v)$. Hence, $\text{dist}_{H_1}(u', v') \leq \text{dist}_{G'}(u', v') + O(\alpha)$. Since $H_2$ is an exact $D$-diameter hopset for $H_1$, we get:

$$\text{dist}^{(D)}_{H_1 \cup H_2}(u', v') = \text{dist}_{H_1}(u', v') \leq \text{dist}_G(u', v') + O(\alpha) \ .$$

Letting $P$ be the shortest $u'$-$v'$ path with at most $D$ hops in $H_1 \cup H_2$, we get that the $u$-$v$ path $P' = (u, u') \circ P \circ (v', v) \subseteq G \cup H$ has at most $O(D)$ hops and of total length $\text{dist}_G(u, v) + O(\alpha)$. The theorem follows. ◀

## References


1. Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Comput. Sci. Rev.*, 37:100253, 2020.
2. Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Stephen G. Kobourov, and Richard Spence. Weighted additive spanners. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science – 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers*, volume 12301 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2020.
3. Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental SSSP in dense weighted digraphs. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1112–1122. IEEE, 2020.
4. Greg Bodwin. New results on linear size distance preservers. *SIAM J. Comput.*, 50(2):662–673, 2021.

5 Greg Bodwin and Gary Hoppenworth. Folklore sampling is optimal for exact hopsets: Confirming the $\sqrt{n}$ barrier. *CoRR*, abs/2304.02193, 2023. doi:10.48550/arXiv.2304.02193.

6 Greg Bodwin, Gary Hoppenworth, and Ohad Trabelsi. Bridge girth: A unifying notion in network design. *CoRR*, abs/2212.11944, 2022. doi:10.48550/arXiv.2212.11944.

7 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Improved weighted additive spanners. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

8 Michael Elkin, Yuval Gitlitz, and Ofer Neiman. Almost shortest paths with near-additive error in weighted graphs. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*, volume 227 of *LIPIcs*, pages 23:1–23:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

9 Jeremy T. Fineman. Nearly work-efficient parallel algorithm for digraph reachability. *SIAM J. Comput.*, 49(5), 2020. doi:10.1137/18M1197850.

10 Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 686–697. IEEE Computer Society, 2018.

11 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Decremental SSSP in weighted digraphs: Faster and against an adaptive adversary. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2542–2561. SIAM, 2020.

12 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved algorithms for decremental single-source reachability on directed graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 725–736. Springer, 2015.

13 William Hesse. Directed graphs requiring large numbers of shortcuts. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 665–669. ACM/SIAM, 2003.

14 Shang-En Huang and Seth Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. *SIAM J. Discret. Math.*, 35(3):2129–2144, 2021. doi:10.1137/19M1306154.

15 Arun Jambulapati, Yang P. Liu, Yang P. Liu, and Aaron Sidford. Parallel reachability in almost linear work and square root depth. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1664–1686. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00098.

16 Philip N. Klein and Sairam Subramanian. A randomized parallel algorithm for single-source shortest paths. *J. Algorithms*, 25(2):205–220, 1997.

17 Shimon Kogan and Merav Parter. Having hope in hops: New spanners, preservers and lower bounds for hopsets. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 766–777. IEEE, 2022.

18 Shimon Kogan and Merav Parter. New diameter-reducing shortcuts and directed hopsets: Breaking the barrier. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 1326–1341. SIAM, 2022.

19 Shimon Kogan and Merav Parter. Faster and unified algorithms for diameter reducing shortcuts and minimum chain covers. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 212–239. SIAM, 2023. doi:10.1137/1.9781611977554.ch9.

20 Shimon Kogan and Merav Parter. New additive emulators. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

21 Kevin Lu, Virginia Vassilevska Williams, Nicole Wein, and Zixuan Xu. Better lower bounds for shortcut sets and additive spanners via an improved alternation product. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 3311–3331. SIAM, 2022.

22 David Peleg and Alejandro A. Schäffer. Graph spanners. *J. Graph Theory*, 13(1):99–116, 1989.

23 Sofya Raskhodnikova. Transitive-closure spanners: A survey. In Oded Goldreich, editor, *Property Testing – Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*, pages 167–196. Springer, 2010. doi:10.1007/978-3-642-16367-8_10.

24 Mikkel Thorup. On shortcutting digraphs. In Ernst W. Mayr, editor, *Graph-Theoretic Concepts in Computer Science, 18th International Workshop, WG '92, Wiesbaden-Naurod, Germany, June 19-20, 1992, Proceedings*, volume 657 of *Lecture Notes in Computer Science*, pages 205–211. Springer, 1992.

25 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.


## A Missing Proofs

**Proof of Lemma 1.7.** Let $TC' = TC(G)[S]$ be the induced transitive closure on the subset $S$. Compute a collection of $O(|S|/D)$ vertex-disjoint paths $\mathcal{P}$ in $TC'$ such that $TC' \setminus V(\mathcal{P})$ has no directed path of length $D/10$. Let $H_1 = \bigcup_{P \in \mathcal{P}} H(P)$ where $H(P)$ is a 2-diameter hopset for $P$ that consists of $O(|P| \log |P|)$ edges, using Lemma 1.5. Let $S' = S[p]$ and $\mathcal{P}' = \mathcal{P}[p]$ for $p = \Theta(\log n/D)$. Let $H_2 = \{e(v, P) \mid v \in S', P \in \mathcal{P}'\}$ where $e(v, P)$ is an edge connecting $v$ to its first outgoing neighbor in $TC'$ on $P$. The size bound is immediate.

Consider the diameter argument. Let $Q$ be a $u$-$v$ shortest path in $G$ for $u, v \in S$ such that $|Q \cap S| \geq D$. By the properties of the paths $\mathcal{P}$, we can assume that $|Q \setminus V(\mathcal{P})| \leq D/10$. We next show that $Q$ can be transformed into a path $Q' \subseteq G \cup H_1$ such that the following holds: (i) for each $P \in \mathcal{P}$, $|P \cap Q'| \leq 3$ and $V(Q') \setminus V(\mathcal{P}) \subseteq V(Q)$. This can be obtained by traversing $Q$ and at each point of observing a vertex $z \in P \cap Q$, we add the shortcut edges $H(P)$ to connect $z$ with the far most vertex $z' \in P \cap Q$. It is easy to see that $V(Q') \subseteq V(Q)$ as by shortcutting $Q$ we can only omit vertices.

Finally, in the case where $|Q' \cap S| \geq D/2$, by property (i), we have that $Q'$ intersects with $\Omega(D)$ distinct paths from $\mathcal{P}$. Let $w \in S$ be the first sampled vertex in $Q' \cap S'$ (w.h.p., such exists among the first $D/10$ many $S$' vertices on $P$). Let $w' \in S$ be the last vertex on $Q'$ that belongs to a sampled path $P'$ in $\mathcal{P}'$ (w.h.p., such exists among that last $D/10$ many $S$' vertices on $P$). The diameter argument holds by noting that the shortcut edge $e(w, P')$ is in $H_2$. ◀

**Proof of Theorem 3.1.** The lower bound graph of [14] is a DAG with $D_G$ layers which contain $\Omega(n)$ critical pairs. For each critical pair $\langle u, v \rangle$, $u$ belongs to layer 1 and $v$ belongs to layer $D_G$. Furthermore, there is a unique directed path in $G$ between $u$ to $v$ and this path contains exactly one vertex from each layer. We now remove the directions of the edges in $G$ to get an undirected and unweighted graph $G'$. Notice that now for each critical pair $\langle u, v \rangle$ in $G'$, we have a unique shortest path of length $D_G - 1$ between $u$ and $v$, that is there might be other paths between $u$ and $v$ but the length of such path will be greater than $D_G - 1$ because such path will necessarily contain at least one vertex from each level, and for a certain level $i$ there will be two vertices from level $i$. In particular, the path will contain

vertices $u, w, v$ consecutively in the path, where $u$ is in level $i$, $w$ is in level $i-1$ and $v$ is in level $i$. We therefore have that the unique directed $u$-$v$ paths in $G$ are translated into unique undirected unweighted shortest paths in $G'$. Therefore, the claim holds in the exact same manner as in [14]. ◀

# Faster Block Tree Construction

**Dominik Köppl**
Department of Computer Science, University of Muenster, Germany

**Florian Kurpicz**
Karlsruhe Institute of Technology, Germany

**Daniel Meyer**
Karlsruhe Institute of Technology, Germany

## Abstract

The block tree [Belazzougui et al. J. Comput. Syst. Sci. '21] is a compressed text index that can answer access (extract a character at a position), rank (number of occurrences of a specified character in a prefix of the text), and select (size of smallest prefix such that a specified character has a specified rank) queries. It requires $O(z \log(n/z))$ words of space, where $z$ is the number of Lempel-Ziv factors of the text. For some highly repetitive inputs, a block tree can require as little as 0.015 bits per character of the text. Small values of $z$ make the block tree a space-efficient alternative to the wavelet tree, which is another index for these three types of queries. While wavelet trees can be constructed fast in practice, up so far compressed versions of the wavelet tree only leverage statistical compression, meaning that they are blind to spaced repetitions.

To make block trees usable in practice, a first step is to find ways in constructing them efficiently. We address this problem by presenting a practically efficient construction algorithm for block trees, which is up to an order of magnitude faster than previous implementations. Additionally, we parallelize our implementation, making it the first block tree construction implementation that works in parallel in shared memory.

**2012 ACM Subject Classification** Theory of computation → Data compression; Theory of computation → Pattern matching

**Keywords and phrases** compressed data structure, block tree, Lempel-Ziv compression, longest previous factor array, rank and select

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.74

**Supplementary Material** *Software (Source Code)*: `https://github.com/pasta-toolbox/block_tree`, archived at `swh:1:dir:534632174136011114d40181f0dcd87e61ddfc4f`
*Software (Comparison with Competitors and Raw Data)*: `https://github.com/pasta-toolbox/block_tree_experiments`, archived at `swh:1:dir:add3d6b114766d0a06532e612ad0f6d08cebdff9`

**Funding** This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).
*Dominik Köppl*: Supported by JSPS KAKENHI Grant Numbers JP21K17701 and JP23H04378.

## 1 Introduction

We experience an every-increasing amount of textual data produced in various domains. Examples include the exponentially increasing capability to sequence genetic data thanks to technical advances [63], code repositories such as GitHub, or natural text collections such as the English Wikipedia, which grows by around 2 million pages each year (currently there are over 58 million pages)[1]. Since there is no expectation that the production of such texts will decelerate, it seems that we start to drown in this sheer amount of data. Nevertheless, for the addressed examples, there is hope in that the produced textual data is usually highly

[1] See `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`, last accessed 2023-07-04.

repetitive: When sequencing two human individuals, we can expect to find that they share more than 99.9 % of genetic data. In other domains such as code repositories or natural text collections, version control systems are used to track all versions of a document or source code to make it possible to revert changes or compare different versions. Since new versions often introduce only small changes, collections of all versions of the same document are often highly repetitive.

When stored or transmitted, texts are oftentimes compressed to save disk space or bandwidth, respectively. The most popular techniques for lossless text compression are based on the Lempel-Ziv 77 (LZ77) factorization [66]. Given $z$ is the number of factors of the LZ77 factorization of a given text, we can represent the text in $O(z)$ words of space. In many use cases, it does not suffice to only store or transmit textual data: the data also has to be processed. A naive way would be to decompress the data before processing it, which is, however, prohibitive for massive datasets. To avoid unnecessary decompression, we can use compressed text indices, which allow us to answer queries efficiently without decompression, while also guaranteeing us (asymptotically) the same space as the compressed text.

The *block tree* [6] is such a compressed text index that requires $O(z \log(n/z))$ words of space for a text $T$ of length $n$ with $z$ LZ77 factors. By default it can answer access queries. However, it can be augmented with additional information to also answer rank and select queries. The queries are defined as follows.

- $\mathrm{access}(T, i)$ returns the character at position $i$, i.e., $T[i]$ for $i \in [0, n)$,
- $\mathrm{rank}_\alpha(T, i)$ returns the number of occurrences of the character $\alpha$ in the $i$-th prefix of the text, i.e., $\mathrm{rank}_\alpha(T, i) = |\{j \leq i \colon T[j] = \alpha\}|$ for $\alpha \in \Sigma$ and $i \in [0, n)$, and
- $\mathrm{select}_\alpha(T, i)$ returns the position of the first character $\alpha$ that has rank $i$, i.e., $\mathrm{select}_\alpha(T, i) = \min\{j \colon \mathrm{rank}_\alpha(T, j) = i\}$ for $\alpha \in \Sigma$ and $i \leq \mathrm{rank}_\alpha(T, n-1)$.

One of the most popular data structures answering all three types of queries is the *wavelet tree* [35]. It is used in, among others, compressed full text indices based on the BWT [23,31,51] or on a grammar [15,16], lossless data compression [22,37,41], and computational geometry [14]. For more related work, see Section 3. Using the block tree, all these queries can be answered in $O(\log(n/z))$ time, with different space-time trade-offs available, see Section 4.

**Our Contribution.** In this paper, we present a block tree construction algorithm that leverages properties of the longest previous factor array, which is a common tool for computing the LZ77 factorization. We analyze our algorithm and show that it has the same asymptotic time complexity as previously presented construction algorithms. However, in our experimental evaluation, we observe that the implementation of our proposed algorithm is up to an order of magnitude faster than previous implementations. Finally, we show that our construction algorithm can also be parallelized.

## 2 Preliminaries

Let $T = T[0]T[1] \dots T[n-1]$ be a text of length $n$ over an alphabet $\Sigma = [0, \sigma)$. The substring $T[i..j] = T[i] \dots T[j]$ is called *prefix* if $i = 0$ and *suffix* if $j = n-1$.

The *Lempel-Ziv 77 (LZ77) factorization* [66] parses the text into $z$ factors $f_0, \dots, f_{z-1} \in \Sigma^+$ such that $T = f_0 \dots f_{z-1}$. For all $i \in [0, z)$, $f_i$ is either a single character not occurring in $f_0, \dots, f_{i-1}$ or the longest substring occurring at least twice in $f_0, \dots, f_i$. The LZ77 factorization can be computed in linear time and space (see Ref. [2] for a survey).

The *longest previous factor array* `LPF` stores at its $i$-th entry the length $\ell$ of the longest substring $T[i..i+\ell)$ having a previous occurrence in the text [17], i.e., $\mathtt{LPF}[i] = \max\{\ell \colon T[i..i+\ell) = T[j..j+\ell) \text{ for } j < i\}$ for $i \in [0, n)$. In particular, if $i$ is the starting position of an

LZ77 factor $f$, then $\texttt{LPF}[i] = |f|$, and thus we can compute the LZ77 factorization in linear time by scanning the LPF array, which can be constructed in linear time [17]. Later on, we also need the position of the occurrence of a longest previous factor, which we store in the *previous occurrence array* PrevOcc. The previous occurrence array is also called quasi suffix array [27]. Here, for all $i \in [0, n)$ we have $T[i..i + \texttt{LPF}[i]) = T[\texttt{PrevOcc}[i]..\texttt{PrevOcc}[i] + \texttt{LPF}[i])$ if $\texttt{LPF}[i] > 0$. We write $\texttt{PrevOcc}[i] = -1$ if $\texttt{LPF}[i] = 0$, i.e., when $T[i]$ is the leftmost occurrence of a single character in $T$.

## 3 Related Work

In this section, we give related work for compressed data structures answering our three types of queries (access, rank, and select) and work on block trees.

### Access, Rank, and Select Data Structures

Answering queries such as access, rank, select are profound problems that have been well addressed in literature. Starting with the case for binary alphabets, there are plenty of results for indexing compressed [10,31,54,55,60] and uncompressed [34,43,47,53,57,64,65] bit vectors. A recent compressed approach involves the linear approximation of the distributions of parts of the ranks in the bit vector [10]. Despite that block trees also work on general alphabets, a block tree variant over the gapped compressed integer array of the ranks of the bit vector can be used to answer rank and select queries [24].

For larger alphabets, we are aware of statistically compressed data structures, where space is expressed in relation to the $k$-th order of empirical entropy $H_k$ with $k = o(\log_\sigma n)$. Most prominent is the Huffman-shaped wavelet tree [35] using $n(H_0(T) + 1) + o(n(H_0(T) + 1)) + O(\sigma \log n)$ bits, and solving all three queries in $O(\log \sigma)$ time. This time could be reduced to $O(1 + \log \sigma / \log \log n)$ with multiary wavelet trees [25], and by a later work [33], the space got reduced to $nH_0(T) + o(n)$ bits. In practice, (Huffman-shaped) wavelet trees are also well-engineered [13, 18, 19, 20, 21, 26, 28, 45].

For faster queries on large alphabets, Golynski et al. [32] gave a data structure taking $n \lg \sigma + o(n \log \sigma)$ bits that answers all three types of queries in $O(\log \log \sigma)$ time. The space got improved by Barbay et al. [4] to $nH_0(T) + o(n(H_0(T) + 1))$ bits. Allowing slightly worse time complexities, Barbay et al. [5] achieved $nH_k(T) + o(n \log \sigma)$ bits, answering all queries in $o((\log \log \sigma)^{1+\epsilon})$ time, for any fixed constant $\epsilon > 0$. These time bounds were improved by Grossi et al. [36] to $O(\log \log \sigma)$ for rank and select, and constant time for access. Finally, Belazzougui and Navarro [9] presented a data structure achieving $nH_k(T) + o(n \log \sigma)$ bits of space, while answering rank in $O(\log \log_w \sigma)$ time. It further can answer access and select in $O(1)$ and any time in $\omega(1)$, respectively, or the other way around. There are also several results on lower bounds for data structures answering the three queries we address here.

Another line of research is to augment grammar compression with an index to support our queries. Here, Belazzougui et al. [7] and Pereira et al. [56] presented grammar indices answering rank and select queries in $O(\log n)$ time. Their indices use $O(g\sigma \log n)$ bits of space when built on a grammar of size $g$, where the latter reference requires that the grammar is balanced. This requirement can be dropped in the light that a grammar can be made balanced in linear time [29].

### Block Trees

Finally, we focus on block trees. Block trees have been proposed by Belazzougui et al. [8], who proposed a Monte Carlo construction algorithm using Karp-Rabin fingerprints in the external memory model. In the journal version [6], the authors provided two construction algorithms which we analyze in Section 4.1. Navarro [50, Section 4.2] recently surveyed block trees, who addresses also most of the references below for applications and variations.

A first application is pattern matching, where Navarro [49] uses block trees for locating pattern in the text. His index uses $O(z \log(n/z))$ words of space and finds all *occ* occurrences of a pattern of length $m$ in $O(m^2 \log n + occ \log^{\epsilon} n)$ time for any constant $\epsilon > 0$. Brisaboa et al. [11] presented an extension of block trees to a two-dimensional data structure simulating $k^2$-trees. Recently, Cáceres and Navarro [12] applied block trees for the compression of the suffix tree topology, the suffix array, and its inverse.

Despite the fact that the space of block trees is related to the size of the LZ77 factorization, the space can, if we vary the definition of a block tree, be made related to the size $\gamma$ of a string attractor [39] or the substring complexity $\delta$ [61].

For the former (string attractor size $\gamma$), we recall that a string attractor is a set of positions of the text such that each substring has an occurrence in the text that contains a position of the string attractor. Kempa and Prezza [39, Theorem 5.3] gave a variant of block trees whose blocks cover substrings of consecutive string attractor positions and thus partition $T$ irregularly. Their variant uses $O(\gamma \log(n/\gamma)) = O(z \log(n/z))$ space and extracts a length-$\ell$ substring in $O(\log(n/\gamma) \cdot (1 + \ell)/\log_\sigma n)$ time. For indexing, Prezza [58] (for rank and select queries) and Navarro and Prezza [52] (for pattern matching) could revert the property that blocks on the same level have equal length while retaining the space size.

For the latter (substring complexity $\delta$), let $\delta := \max\{d_k/k : k \in [1, n]\}$ denote the substring complexity of $T$, where $d_k$ is the number of distinct length-$k$ substrings of $T$. Then there is a block tree variant that can be represented in $O(\delta\tau \log_\tau \frac{n}{\delta})$ space supporting queries in $O(\log \frac{n}{\delta})$ time [40].

## 4 Block Trees

Let $T$ be a text of length $n$ over an alphabet of size $\sigma$, whose LZ77 factorization consists of $z$ factors. The *block tree* [6] is a compressed index requiring $O(z \log(n/z))$ words of space. It supports access, rank, and select queries in $O(\log(n/z))$ time. In the following, we describe a block tree with two integer parameters $s$ and $\tau$ that are greater than 1, which specify the out-degree of the root and all other internal nodes, respectively. For simplicity, we assume that $n = s \cdot \tau^h$ for some integer $h$. Now, the block tree is a tree of height $h = 1 + \log_\tau \frac{z \log n}{s \log n}$ with parameters $\tau$ and $s$ such that the root has $s$ children and every internal node is a leaf or has $\tau$ children.

Each node $u$ represents a substring of $T$ called *block* $B^u$. The root represents the whole text $T$ and has $s$ children representing $s$ consecutive blocks of length $n/s$. We refer to all blocks with the same depth as a *block tree level*. Two *blocks are consecutive* if they are in the same block tree level and if they are consecutive in $T$. Let $B_i \cdot B_{i+1}$ be the concatenated substring of two consecutive blocks. We *mark* the blocks $i$ and $i + 1$, if $B_i \cdot B_{i+1}$ is the leftmost occurrence of that substring in $T$. All non-root nodes that are not in the last level represent either marked or unmarked blocks.

All marked blocks $B^v$ are internal nodes with $\tau$ children. These children represent consecutive blocks of length $|B^v|/\tau$ whose concatenation is $B^v$. Unmarked blocks $B^u$, on the other hand, are leaves that only store a pointer towards the pair of consecutive blocks $B_i \cdot B_{i+1}$ containing the leftmost occurrence of $B^u$ and the offset of that occurrence in the blocks. The number of blocks per level of the block tree is bounded.

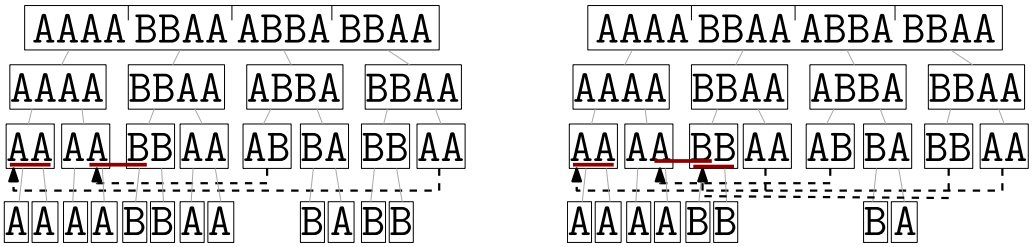



**Figure 1** The block tree (left) and its pruned version (right) for the text $T = \texttt{AAAABBAAABABBAA}$ with $\tau = 2$ and $s = 4$. Dashed arrows indicate pointers to the leftmost occurrence of the block the arrow starts at. The red line indicates the offset stored in addition to the pointer. Note that only characters in leaves are stored explicitly. For simplicity, our leaves contain only a single character. In the pruned block tree, we can replace an additional `AA` block. Note that we cannot prune the second `AA` block, as another block points into it.

▶ **Lemma 1** ([6, Lemma 1]). *Any level of a block tree (except the first) contains* $\leq 3z\tau$ *blocks.*

We have reached the last (or deepest) level of the block tree when explicitly storing the representing substring requires less space than storing the pointer for an unmarked block. At this level, we store the substring of each unmarked block explicitly. For example, if $|B^u| \in \Theta(\log_\sigma n)$, its encoding requires $O(1)$ words of space. Note that on each level, the block length decreases by a factor of $\tau$.

The block tree requires $O(s + z\tau \log_\tau \frac{n \log \sigma}{s \log n})$ words of space. Choosing $\tau$ as constant yields the minimum space requirement of the block tree mentioned above. Choosing $s = z$ results in block trees of size $O(z\tau \log_\tau \frac{n \log \sigma}{z \log n})$ words. Different values for $\tau$ can introduce other space-time trade-offs, as described by Belazzougui et al. [6].

## 4.1 Construction

Belazzougui et al. [6] give two construction algorithms, which we briefly review. Their first algorithm requires $O(n)$ words of working space, where the idea is to use an Aho-Corasick automaton [1] that can identify all consecutive pairs of blocks $B_0 \cdot B_1, B_1 \cdot B_2, \ldots, B_{s-2} \cdot B_{s-1}$ on the first level. This automaton is then used to identify the first occurrences of all pairs and to mark them accordingly. To set the leftwards pointers into unmarked blocks, the automaton is replaced by a new automaton that recognizes all unmarked blocks is created. The text is traversed using this automaton. Whenever an unmarked block is found for the first time, a pointer (and offset) is stored. For then on, the second automaton is no longer of use and can be removed. Subsequently, the algorithm continues with the next level, considering only the unmarked blocks from the previous level.

Their second algorithm uses $O(s + z\tau)$ words of working space and runs in $O(n)$ expected time. Here, the general idea is to replace the Aho-Corasick automaton with Karp-Rabin fingerprints [38], i.e., storing Karp-Rabin fingerprints of all consecutive pairs of blocks $B_i \cdot B_{i+1}$ in a hash table. Since there are at most $3z\tau$ blocks per level, this approach requires only $O(s + z\tau)$ words of working space. Both algorithms work in linear time if $s = \Theta(z)$.

The block tree as described here only supports access queries. For rank and select support, additional information has to be stored for each marked block. In the case of rank queries, the occurrence of all characters in the text up to the beginning of the block is necessary. For more details, we refer to the original block tree paper [6].

**Pruning.** When we construct block trees with one of the two aforementioned construction algorithms, we meet the asymptotic space bounds – but the block tree may contain more blocks than necessary, see Figure 1. Remember that we mark the first occurrence of each pair of consecutive blocks $B_i \cdot B_{i+1}$ to guarantee that any block $B^u$ to their right can point to them. However, there may be no rightwards block pointing to either (or both) $B_i, B_{i+1}$. In this case we would replace one of the blocks (or both) with leftward pointers, if one (or both) of them occurs previously. Since we do not modify this part of the algorithm, we refer to Belazzougui et al.'s [6, Section 6.1] description of the pruning step for more details.

# 5 Block Tree Construction using the LPF Array

We now describe our new block tree construction algorithm based on the LPF array.[2] First, in Section 5.1, we mark blocks using only the LPF array. Then, in Section 5.2, we find the leftmost occurrences of unmarked blocks, before, in Section 5.3, we combine all these ideas to our new algorithm.

## 5.1 Marking Blocks

The block tree is closely related to the LZ77 factorization of the text. Consecutive blocks $B_{i-1}$, $B_i$, and $B_{i+1}$ are only marked if an LZ77 factor starts in $B_i$, i.e., if they contain the leftmost occurrence of some substring. Similarly, LPF values witness the shortest substring starting at each text position that is a leftmost occurrence. Hence, we can make use of the LPF array to mark blocks. Let $\mathtt{s}(B^u)$ denote the starting *text* position of the substring represented by $B^u$.

▶ **Lemma 2.** *Given three consecutive blocks $B_{i-1}, B_i$, and $B_{i+1}$ of length $\ell$. We mark $B_i$ if $\mathtt{LPF}[\mathtt{s}(B_{i-1})] < 2 \cdot \ell$ or $\mathtt{LPF}[\mathtt{s}(B_i)] < 2 \cdot \ell$ is true.*

**Proof.** By the definition of the LPF array, a substring $T[i..i+\ell)$ has a preceding occurrence in the text if $\mathtt{LPF}[i] \geq \ell$. We only leave $B_i$ *un*marked if both $\mathtt{LPF}[\mathtt{s}(B_{i-1})]$ and $\mathtt{LPF}[\mathtt{s}(B_i)]$ are at least $2 \cdot |B_i|$ because this means that there is a previous occurrence of both $B_{i-1} \cdot B_i$, and $B_i \cdot B_{i+1}$. Otherwise, if there is no previous occurrence of one of the two pairs, we have to mark $B_i$. ◀

To determine whether the last block is marked, only its preceding block has to be considered. Otherwise, it is the same argument as used in Lemma 2. Since each level of the block tree contains $O(z\tau)$ blocks, we get the following result.

▶ **Lemma 3.** *Given the LPF array, we can mark all blocks of a level in the block tree in $O(z\tau)$ time.*

## 5.2 Identifying Leftmost Occurrences

Now, for each *unmarked* block, we have to identify the leftmost substring in the text that is equal to that block, as we need to add pointers (and offsets) from the unmarked blocks to these occurrences. Note that in all levels but the first one, the index of the block $B^u$ does not automatically translate to the block's starting position $\mathtt{s}(B^u)$, as we do not know how many blocks have been unmarked to its parent's left in the previous level. Therefore, we need to store additional information regarding a block's starting position for each block.

[2] The description is based on and has text overlaps with Daniel Meyer's Master's thesis [48].

### 5.2.1 Leftmost Occurrences as Text Positions

In this section, we describe how to identify the text position of the previous occurrence. Afterwards, in Section 5.2.2, we show how to identify the block that contains this text position on the current level. While the LPF array is sufficient to mark blocks, it does not contain information necessary to find the leftmost occurrences of blocks that we require for the leftward pointers. We now give a naive approach to compute the text positions in Section 5.2.1.1. Then, in Section 5.2.1.2, we improve the naive approach by using dynamic programming to obtain better asymptotic running times. The general idea in both cases is to follow the leftmost occurrences of previous occurrences for all blocks on a level.

▶ **Lemma 4.** *Let $i \in [0,n)$ be a text position, $j = \texttt{PrevOcc}[i]$, and $k = \texttt{PrevOcc}[j]$. If $0 < \texttt{LPF}[i] \leq \texttt{LPF}[j]$, then $T[i..i+\texttt{LPF}[i]) = T[j..j+\texttt{LPF}[i]) = T[k..k+\texttt{LPF}[i])$.*

**Proof.** $\texttt{LPF}[i] = \max\{k \colon T[i..i+k) = T[j..j+k) \text{ for } j < i\}$ and $\texttt{PrevOcc}[i]$ gives us the position $j$, where this longest factor occurs. Since $0 < \texttt{LPF}[i]$, we have $T[i..i+\texttt{LPF}[i]) = T[j..j+\texttt{LPF}[i])$ by definition. We also know that for text position $j$, there exists a previous factor of length at least $\texttt{LPF}[i]$ at position $k$. Hence $T[i..i+\texttt{LPF}[i]) = T[k..k+\texttt{LPF}[i])$. ◀

The same holds not only for length-$\texttt{LPF}[i]$ substrings, but for general length-$\ell$ substrings with $\ell \leq \texttt{LPF}[i]$, which is useful when processing a block tree level where blocks have all the same length $\ell$.

▶ **Observation 5.** *Let $i \in [0,n)$ be a text position and $j = \texttt{PrevOcc}[i]$. If $0 < \ell \leq \texttt{LPF}[i]$, then $T[i..i+\ell) = T[j..j+\ell)$.*

#### 5.2.1.1 Naive Approach

Using these properties, we can describe a *naive* algorithm to find the leftmost occurrence of a given unmarked block $B^u = T[i..i+\ell)$. Here, we simply follow the PrevOcc entries until the length of the longest previous factor of the previous occurrence is smaller than $\ell$. This leads us to the first occurrence of length $\ell$. Unfortunately, this naive approach requires $O(n)$ time for each block. For example, in an all-a text aa...a, for text position $i$ we would follow the longest previous occurrences $i-1, i-2, \ldots, 0$ in case that $\texttt{PrevOcc}(i) = i-1$ for all $i > 0$. Therefore, using the naive approach, we do not achieve the asymptotic running time of the original block tree construction algorithms by Belazzougui et al. [6]. However, in practice, this approach works very well, as we observed in our experimental evaluation in Section 6.

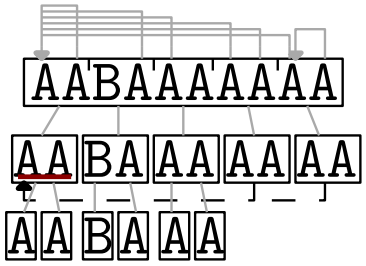
#### 5.2.1.2 Dynamic Programming

To retain the time complexities of the original block tree construction algorithms, we make use of dynamic programming to find the leftmost occurrences of a block in the text. We start with the definitions of $\ell$-factors and $\texttt{FirstOcc}_\ell$.

▶ **Definition 6** ($\ell$-factor and $\texttt{FirstOcc}_\ell$). *For $\ell > 0$, we denote $T[i..i+\ell)$ as $\ell$-factor$_i$. $\texttt{FirstOcc}_\ell[i]$ stores $\texttt{PrevOcc}[i]$, if no previous occurrence of $\ell$-factor$_i$ exists (remember that $\texttt{PrevOcc}[i] = -1$ if $T[i]$ is the leftmost occurrences of a character, i.e., $\texttt{LPF}[i] = 0$) and the leftmost occurrence of $\ell$-factor$_i$ otherwise.*

We can compute $\texttt{FirstOcc}_\ell$ using *dynamic programming* by iterating over PrevOcc from left to right. To start with, we set $\texttt{FirstOcc}_\ell[0] = -1$. Suppose that we have processed $\texttt{FirstOcc}_\ell[0..i-1]$ and are at text position $i$. For $j := \texttt{PrevOcc}[i]$, we consider two cases:

**Table 1** LPF, PrevOcc, $\texttt{FirstOcc}_2$, and the block tree (with parameters $s = 5, \tau = 2$) for the string AABAAAAAAA. The arrows above the first level of the block tree indicate the $\texttt{FirstOcc}_2$ values.

| $i$ | $T[i]$ | $T[i..n)$ | LPF$[i]$ | PrevOcc$[i]$ | $\texttt{FirstOcc}_2[i]$ | block tree |
|---|---|---|---|---|---|---|
| 0 | A | AABAAAAAAA | 0 | -1 | -1 |  |
| 1 | A | ABAAAAAAA | 1 | 0 | 0 | |
| 2 | B | BAAAAAAA | 0 | -1 | -1 | |
| 3 | A | AAAAAAA | 2 | 0 | 0 | |
| 4 | A | AAAAAA | 6 | 3 | 0 | |
| 5 | A | AAAAA | 5 | 4 | 0 | |
| 6 | A | AAAA | 4 | 5 | 0 | |
| 7 | A | AAA | 3 | 6 | 0 | |
| 8 | A | AA | 2 | 7 | 0 | |
| 9 | A | A | 1 | 8 | 8 | |

**Case 1** $\texttt{LPF}[i] \geq \ell$ and $\texttt{LPF}[j] \geq \ell$: From $\texttt{LPF}[j] \geq \ell$ follows that $\ell$-factor$_j$ has a previous occurrence. Combined with Observation 5 and $\texttt{LPF}[i] \geq \ell$, we can conclude that the previous occurrence of $\ell$-factor$_j$ is also an occurrence of $\ell$-factor$_i$. As we already calculated $\texttt{FirstOcc}_\ell[j]$, we can set $\texttt{FirstOcc}_\ell[i] = \texttt{FirstOcc}_\ell[j]$.

**Case 2** $\texttt{LPF}[i] < \ell$ or $\texttt{LPF}[j] < \ell$: We set $\texttt{FirstOcc}_\ell[i] = j$ since either $T[i..i+\ell)$ or $T[j..j+\ell)$ is the leftmost occurrence of $\ell$-factor$_i$.

See Table 1 for an example. Note that we still need the LPF array to correctly interpret any $\texttt{FirstOcc}_\ell[i]$. For $\texttt{LPF}[i] \geq \ell$ but $\texttt{LPF}[j] < \ell$ we know that $\ell$-factor$_i$ has an earlier occurrence at $j$ but no occurrence further left. This dynamic programming approach requires $O(n)$ time to compute $\texttt{FirstOcc}_\ell$. Such time is unfeasible if we need to calculate $\texttt{FirstOcc}_\ell$ for each level of the block tree.

However, it is possible to compute $\texttt{FirstOcc}_{\ell_0}$ using $\texttt{FirstOcc}_{\ell_1}$ for $\ell_1 \geq \ell_0$, as every occurrence of an $\ell_1$-factor contains an $\ell_0$-factor as a prefix. There might be an occurrence of the $\ell_0$-factor further left, but we store information about that in $\texttt{FirstOcc}_{\ell_1}$. Remember that we also store pointers to a previous occurrence of the longest previous factor if that factor is shorter than $\ell_1$.

We can now use this property to identify the leftmost occurrence for each block level-by-level in $\texttt{FirstOcc}_*$, where we store $\texttt{FirstOcc}_\ell$ for the current level and update it for each following level. Recall that by definition, each pair of marked blocks contains the leftmost occurrence of at least one substring of $T$. Therefore, each leftmost occurrence of any substring with length at most the current block level length $\ell$ is contained in a marked block. All blocks in the current level (except for the first level) are children of marked blocks in the level before. Hence, the leftmost occurrence of each substring of a length equal to the current block length is contained in a block in the current level. We still have to update all text positions contained in the previous block tree level. This is necessary as we need to consider cases where the values in the LPF array are smaller than the last level's block size $\ell_1$ but greater than the next level's block size $\ell_0$, i.e., the positions $i \in [0, n)$ where $\ell_0 \leq \texttt{LPF}[i] < \ell_1$.

In this case $\texttt{FirstOcc}_{\ell_1}[i]$ points to a previous occurrence of the longest previous factor of $i$. This occurrence can be in an unmarked block as it is not necessarily the first occurrence of said longest previous factor. Hence, we also have to update $\texttt{FirstOcc}_*$ for text positions $k \in [0, n)$ that fall into an unmarked block in the previous level. We do so from left to right. Suppose we have updated $\texttt{FirstOcc}_*[0..k-1]$ and are processing $\texttt{FirstOcc}_*[k]$. Let $p = \texttt{FirstOcc}_*[k]$. We will update $\texttt{FirstOcc}_*[k]$ if one of the two conditions is met.

**Condition 1** $\texttt{LPF}[k] \geq \ell_0$ and $\texttt{LPF}[p] \geq \ell_0$: We know that $p$ and $k$ share the same $\ell_0$-factor (Observation 5). Therefore, the first occurrence of $\ell_0$-factor$_p$ is also the first occurrence of $\ell$-factor$_k$, and we can set $\texttt{FirstOcc}_*[k] = \texttt{FirstOcc}_*[p]$.

**Condition 2** $0 < \texttt{LPF}[k] \leq \texttt{LPF}[p]$: If condition 2 is met but condition 1 is not, we still know that $\texttt{FirstOcc}_*[k] = \texttt{FirstOcc}_*[p]$, as there are just two cases:

**Case 2.1** $\texttt{LPF}[k] < \ell_0$ and $\texttt{LPF}[p] \geq \ell_0$: Due to condition 2 and Lemma 4 and Observation 5 we know that said longest previous factor is a prefix of $\ell_0$-factor$_p$. Due to $\texttt{LPF}[p] \geq \ell_0$, $\texttt{FirstOcc}_*[p]$ points to the leftmost occurrence of $\ell_0$-factor$_p$.

**Case 2.2** $\texttt{LPF}[k] < \ell_0$ and $\texttt{LPF}[p] < \ell_0$: Due to condition 2 and Lemma 4 we know that said longest previous factor has a previous occurrence at $\texttt{FirstOcc}_*[p]$.

If neither condition is met, i.e., $\texttt{LPF}[p] < \texttt{LPF}[k]$ and $\texttt{LPF}[p] < \ell_0$, $k$ is either the leftmost occurrence for all factors of size at least $\ell_0$ or $\texttt{FirstOcc}_*[k]$ points to a first occurrence of a substring smaller than $\ell_0$ and hence points into a marked block.

▶ **Lemma 7.** *Updating* $\texttt{FirstOcc}_*$ *for all levels during the block tree construction requires* $O(n(1 + \log_\tau \frac{z}{s}))$ *time in total.*

**Proof.** Initializing $\texttt{FirstOcc}_*$ takes $O(n)$ time. Every level but the first has at most $3z\tau$ blocks, and the size of blocks is decreasing by a factor $\tau$ for each further level. This reduces the number of string positions still contained inside of blocks geometrically with each level. The total sum of all block lengths is $O(n(1 + \log_\tau \frac{z}{s}))$ [6, Section 6.1]. Since we only have to update $\texttt{FirstOcc}_*$ for positions in unmarked blocks in the previous level, we obtain the required total time for all levels. ◀

### 5.2.2 Leftmost Occurrences as Blocks

After updating $\texttt{FirstOcc}_*$ to store the leftmost occurrence for each text position in the current block tree level $B_0, B_1, \ldots$, we still have to find the marked blocks covering the leftmost occurrence of each unmarked block $B^u$.

We can map the occurrences in the text to blocks in three parts. First, we store for each unmarked block $B_i$ a pair $\langle \texttt{FirstOcc}_*[\texttt{s}(B_i)], i\rangle$ containing its leftmost occurrence and its index in our block level in a set $U$. Second, we sort the set by each pair's first element using a radix sort, i.e., we sort our unmarked blocks by their leftmost occurrences in the text. Third, we sequentially scan our block tree level and our sorted set $U$ simultaneously in the following fashion. Let $\langle occ_j, j\rangle$ be the currently considered element in $U$ and $B_i$ be the current block of our block tree level. If $\ell$-factor$_{occ_j}$ starts inside $B_i$, we set a pointer from $B_j$ to $B_i$ with offset $occ_j - \texttt{s}(B_i)$ and continue with the next element in $U$. Otherwise, we continue with the next block in our block tree level.

▶ **Lemma 8.** *For block trees with* $z\tau = O(n)$*, finding the blocks containing the leftmost occurrences of unmarked blocks can be done in* $O(z\tau)$ *time and* $O(z\tau)$ *words of space.*

**Proof.** The first and third step require $O(z\tau)$ time, as we only scan over blocks in the current block tree level. Sorting $U$ with radix sort requires $O(z\tau)$ time and $O(z\tau)$ words of space. ◀

Alternatively, we can also identify the mapping between text positions and blocks by traversing the already built block tree. This mimics an access query on the block tree that stops as soon as it reaches the current level. Such a query requires $O(\log_\tau \frac{n \log \sigma}{s \log n})$ time, which is linear in the height of the tree. Overall, computing the mapping using this approach requires $O(z\tau \log_\tau \frac{n \log \sigma}{s \log n})$ time.

**Table 2** Input names, number of characters $n$, alphabet size $\sigma$, number of LZ77 factors $z$, measure of compressibility $\frac{z \log n}{n \log \sigma}$, and the compression achieved with `p7zip` (v. 16.02) expressed by the ratio of the compressed output size divided by the input size.

| | Input | $n$ | $\sigma$ | $z$ | $\frac{z \log n}{n \log \sigma}$ | `p7zip` |
|---|---|---|---|---|---|---|
| repetitive | cere | 461 286 644 | 5 | 1 700 630 | 0.044 | 5.35 % |
| | coreutils | 205 281 778 | 236 | 793 915 | 0.013 | 11.75 % |
| | einstein.en | 467 626 544 | 139 | 89 467 | 0.0007 | 0.10 % |
| | Escherichia_coli | 112 689 515 | 15 | 2 078 512 | 0.121 | 7.76 % |
| | influenza | 154 808 555 | 15 | 769 286 | 0.033 | 1.69 % |
| | kernel | 257 961 616 | 160 | 1 446 468 | 0.021 | 2.53 % |
| | para | 429 265 758 | 5 | 2 332 657 | 0.064 | 6.05 % |
| | world_leaders | 46 968 181 | 89 | 175 740 | 0.014 | 1.39 % |
| non-repetitive | dblp.xml | 296 135 874 | 97 | 9 576 081 | 0.138 | 12.74 % |
| | dna | 403 927 746 | 16 | 25 628 189 | 0.453 | 22.79 % |
| | english | 1 610 612 736 | 239 | 97 047 354 | 0.233 | 26.11 % |
| | pitches | 55 832 855 | 133 | 5 994 276 | 0.391 | 25.89 % |
| | proteins | 1 184 051 855 | 27 | 80 408 252 | 0.430 | 31.30 % |
| | sources | 210 866 607 | 230 | 11 598 459 | 0.194 | 15.84 % |

## 5.3 New Block Tree Construction Algorithm

Now, we can put all these building blocks together to form a practically efficient block tree construction algorithm. First, we calculate the `LPF` array and `PrevOcc` array. We then initialize $\texttt{FirstOcc}_*$ for $\ell = n/s$ and construct each block tree level top-down as described in this section, i.e., we identify all marked blocks and then compute all pointers (and offsets) for the unmarked blocks. Finally, we update $\texttt{FirstOcc}_*$ and repeat this process until we reach the deepest level, where the blocks are stored explicitly. While this algorithm does not introduce better asymptotic running times, it practically outpaces all other available construction implementations, as we will empirically evaluate in Section 6. Overall, combining all previous steps, we obtain the following result.

▶ **Theorem 9.** *Given a string $T$ of length $n$ over an alphabet of size $\sigma$, two integers $s$ and $\tau$ greater than 1, we can compute the block tree in $O(n(1 + \log_\tau \frac{z}{s}))$ time using $O(n)$ words of space.*

**Pruning.** The algorithm described in this section constructs the same block tree structure as the algorithms described by Belazzougui et al. [6], see Section 4.1. Thus, their pruning algorithm works without any changes of the block tree resulting from this construction.

# 6 Experimental Evaluation

We conducted our experiments on a server equipped with an AMD EPYC Rome 7702P (64 cores (128 hyperthreads), frequencies up to 3.35 GHz, and 256 MiB L3 cache) and 1024 GiB DDR4 ECC RAM. The server runs Ubuntu 20.04.2 LTS. We compiled all code with GCC 12.1 using the flags `-O3` and `-march=native`. For the evaluation of our parallel code written in OpenMP, we compiled the code with the additional flag `-fopenmp`.

**Table 3** Most space-efficient ($\tau_{\text{space}}$ and $b_{\text{space}}$) and fastest configurations ($\tau_{\text{time}}$ and $b_{\text{time}}$) of $\text{LPF}_1$ on the repetitive inputs (without rank and select support).

| Input | $\tau_{\text{space}}$ | $b_{\text{space}}$ | $\tau_{\text{time}}$ | $b_{\text{time}}$ |
|---|---|---|---|---|
| cere | 16 | 16 | 2 | 8 |
| coreutils | 8 | 8 | 2 | 2 |
| einstein.en.txt | 4 | 16 | 2 | 2 |
| Escherichia_Coli | 4 | 16 | 2 | 4 |
| influenza | 4 | 16 | 2 | 4 |
| kernel | 8 | 16 | 2 | 2 |
| para | 4 | 16 | 2 | 8 |
| world_leaders | 4 | 16 | 2 | 2 |

We compare our block tree construction algorithms [42,44] with the (to our best knowledge) only other block tree implementation by Belazzougui et al. [6].[3] Their implementation uses Karp-Rabin fingerprints (Section 4) with parameter $s = 1$. While $s = 1$ is not a feasible choice in the formal definition of block trees (see Section 4), in practice, all levels without any unmarked blocks are removed during the pruning phase, making this configuration possible.

There are two different variants of our block tree construction algorithm: $\text{LPF}_s^{\text{DP}}$ and $\text{LPF}_s$. The former uses the dynamic programming approach to identify the text position of the previous occurrence while the latter uses the naive approach, see Section 5.2. Since we need the `LPF` array for our construction algorithms, we can compute $z$ and also choose both $s = z$ and $s = 1$. To show that our improvements are not only based on engineering, we also include $\text{FP}_1$ which uses fingerprints instead of the `LPF` array while the rest of our code remains nearly unchanged, i.e., it is a reimplementation of the original algorithm.

For our implementation, we make use of `libsais`[4], the fastest suffix array construction algorithm implementation available to compute the suffix and longest common prefix arrays that we use for the `LPF` array construction. We also use the `int_vector` from the Succinct Data Structure Library [30] and the `pasta::bit_vector` [43] internally in the block tree.

We do not include fast wavelet tree construction algorithms [18, 30] in our plots as preliminary experiments show that they can be constructed at least an order of magnitude faster than block trees. For a detailed comparison of query speed of block trees and wavelet trees, we refer to the block tree article by Belazzougui et al. [6]. They show that block trees require around the same space but can answer queries an order of magnitude faster.

We conducted our sequential experiments with all combinations of $\tau \in \{2, 4, 8, 16\}$ and maximum leaf size $b = \{2, 4, 8, 16\}$, i.e., the threshold on the number of characters stored explicitly as a leaf. The timing starts when the input is loaded in main memory and stops as soon as the block tree has been constructed. All reported values are the average of three runs. We used the repetitive text corpus from the *Pizza&Chili* corpus[5], which was also used in the original block tree article [6]. Additionally, we used the non-repetitive *Pizza&Chili* corpus[6]. See Table 2 for details.

[3] See `https://github.com/elarielcl/BlockTrees`, last accessed 2023-07-04
[4] See `https://github.com/IlyaGrebnov/libsais`, last accessed 2023-07-04.
[5] See `http://pizzachili.dcc.uchile.cl/repcorpus`, last accessed 2023-07-04.
[6] See `http://pizzachili.dcc.uchile.cl/texts.html`, last accessed 2023-07-04.

## 6.1 Sequential Block Tree Construction

In this section, we present the results of our experimental evaluation of block tree construction algorithms. For the evaluation, we used both, repetitive and non-repetitive inputs.

**Repetitive Inputs.** In Figures 2 and 3, we show the construction throughput (processed input in MiB per second) on the $y$-axis and the space requirements of the final block tree (without and with additional rank and select support) on the $x$-axis. Plotting the throughput helps normalizing the running times for different input sizes. Furthermore, it highlights that the construction time of the block tree without rank and select support does not depend on the compressibility of the input.

Surprisingly, on most inputs, smaller block trees are not that much slower to construct than larger block trees. Our fastest construction algorithm for the smallest block trees is always $\mathrm{LPF}_1$. Hence, we now only compare this algorithm with the original implementation. This also means that following the previous occurrence in the text naively is (for small block trees with fewer marked nodes) cheaper than explicitly computing the results. Furthermore, choosing $s = z$ provides no real benefit, as it does not result in a faster construction. For most inputs, the most space-efficient configuration of $\mathrm{LPF}_1$ uses $\tau = 4$ and $b = 16$ and the fastest configuration of $\mathrm{LPF}_1$ uses $\tau = 2$ and $b = 2$, see Table 3. Note that no two different configurations result in the same space requirements, even though, they can be very close.

When constructing only the block tree without rank and select support, $\mathrm{LPF}_1$ is between 6.48 and 11.52 times faster than the original implementation (average: 9.51, median: 9.86). Computing the additional data for the rank and select support is the same for our and the original implementation. Thus, here $\mathrm{LPF}_1$ is only between 3.61 and 11.23 times faster (average: 6.75, median: 6.24), despite the fact that the times for $\mathrm{LPF}_1$ include also the `LPF` array construction. We further want to mention that the `LPF` array construction also introduces higher memory requirements during the construction than the fingerprint-based approaches. Hence, there is a working-space-time trade-off for the construction.

**Non-Repetitive Inputs.** We now give additional experimental results on the non-repetitive inputs. We only use 32 MiB prefixes of the texts, as the block tree construction algorithm by Belazzougui et al. [6] requires more than 1 TiB of working space for larger non-repetitive inputs. This is due to the order in which their algorithm constructs the block tree. Instead of first compressing all data internally, many operations and auxiliary data is computed on the uncompressed data. The results of these experiments are depicted in Figure 5.

Overall, the results are similar to the results for the repetitive inputs: $\mathrm{LPF}_1$ is the fastest construction algorithm most of the time. However, on some inputs, the dynamic programming $\mathrm{LPF}_z^{\mathrm{DP}}$ is faster. This is due to the long chains of previous occurrences that have been marked. Since the texts are non-repetitive, there are fewer marked blocks overall, resulting in bigger block trees. Overall, for non-repetitive inputs, computing larger block trees is slightly faster than computing very space-efficient block trees. This becomes very apparent for block trees with rank and select support.

## 6.2 Parallel Block Tree Construction

While the total running time of our algorithm is fast compared to our competitor, on average 71.33 % of the running time is spent for the `LPF` array construction. Fortunately, `libsais` supports parallel computation. In addition, we use the LZ77 factorization algorithm by

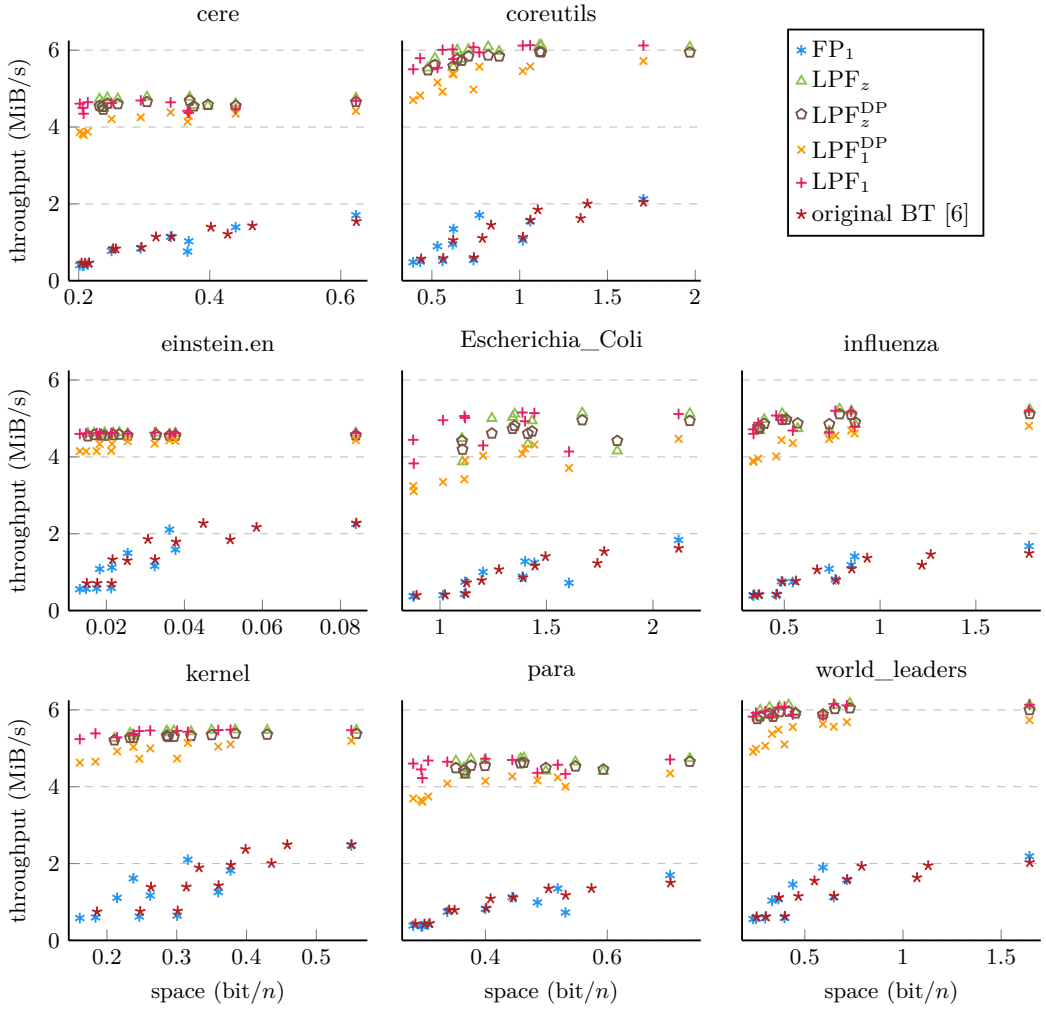



**Figure 2** Block tree construction *without* rank and select support, showing throughput (processed input in MiB per second) and space requirements of the final block tree (bits per character of the input) on repetitive inputs. The range of each $x$-axis depends on the compressibility of the input. Data points for each algorithm show different configurations of $\tau$ and $b$, see Table 3 for more details.

Shun and Zhao [62] that requires $O(n)$ work and $O(\log^2 n)$ time.[7] We also parallelized the construction of the rank and select support with a straight-forward implementation since the computed values are independent for each character.

We only achieve a speedup using up to 32 cores. This is most likely due to the fact that only eight memory controllers are available, which have to be shared by 16 groups of 4 cores. As soon as we use more than 32 cores, multiple groups have to share a controller. With 32 cores, we achieve a speedup of up to 6.64 (4.71 on average). This comes very close to the speedups of the parallel `libsais`, which achieves a speedup of at most 6 (5.2 on average). The additional speedup can be explained by the speedup thanks to the parallel construction of the rank and select support.

[7] See `https://github.com/zfy0701/Parallel-LZ77`, last accessed 2023-07-04.

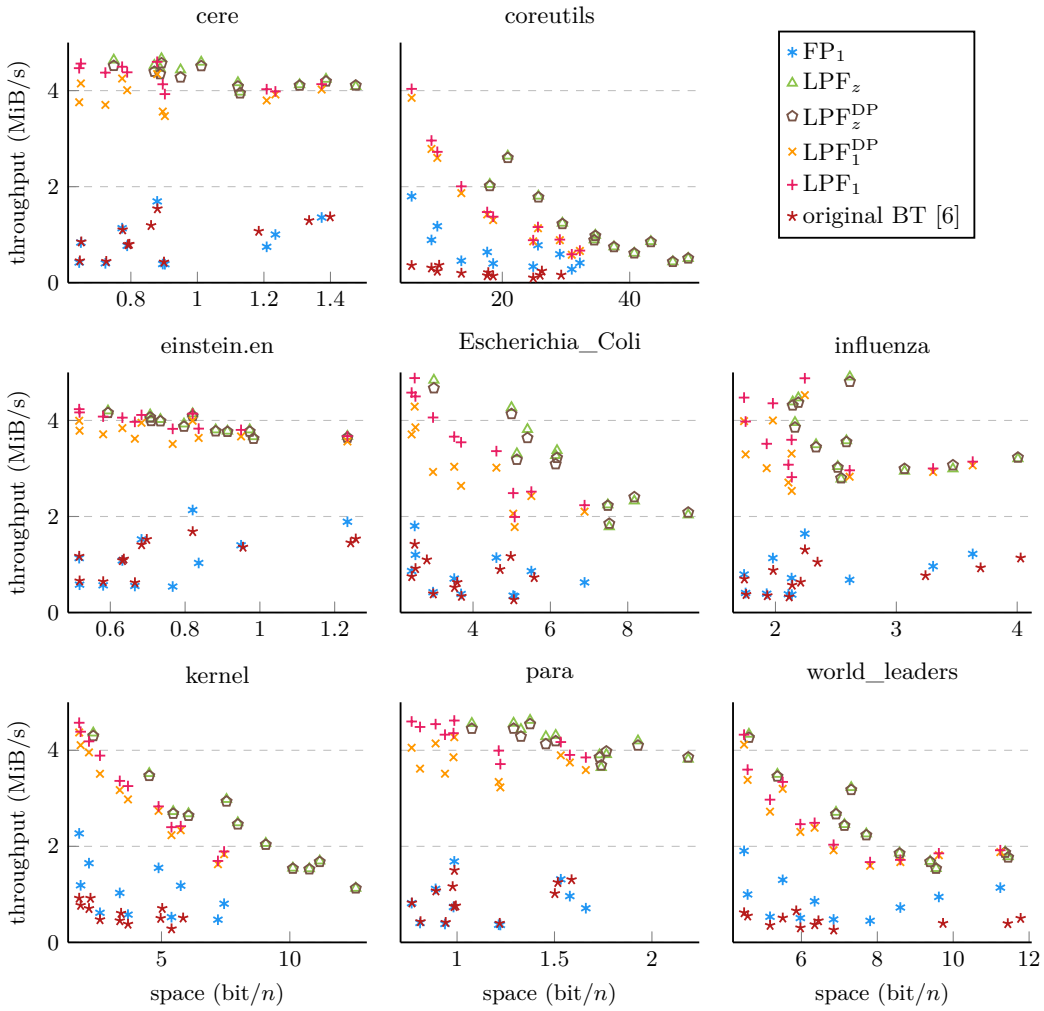



**Figure 3** Block tree *with* rank and select support construction throughput (processed input in MiB per second) and space requirements of the final block tree (bits per character of the input) on repetitive inputs. The range of each $x$-axis depends on the compressibility of the input. Data points for each algorithm show different configurations of $\tau$ and $b$, see Table 3 for more details.

## 7 Conclusion and Future Work

The LPF array allows us to construct the block tree up to an order of magnitude faster than using Karp-Rabin fingerprints. All tested algorithms produce the same block trees (when using the same parameters). A simple parallelization of our algorithm results in a speedup of up to 6.64 using 32 cores. However, the scalability of the current state of the algorithm is mostly limited by the LPF array computation. Here, it might be interesting to investigate a parallelization of the construction algorithm based on Karp-Rabin fingerprints using concurrent hash tables [46]. In general, better scalability is of great interest, as otherwise, construction speed similar to wavelet trees seems hard to achieve.

In the light that the LPF array can be represented in $2n + o(n)$ bits [3] with algorithms computing this representation in compact [3] or compressed space [59], future work includes engineering a more memory-efficient LPF array construction. Further improvements in

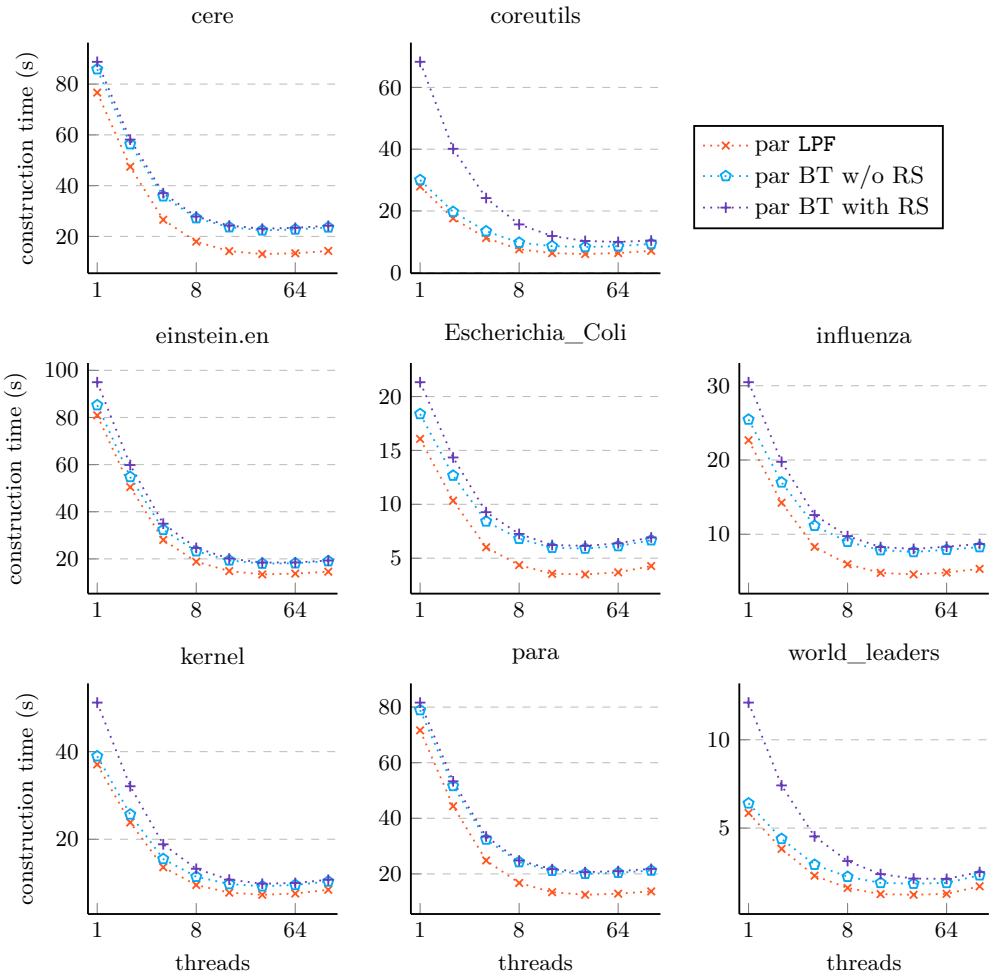



**Figure 4** Parallel (strong scaling) block tree construction using the configuration $\tau = 8$ and $b = 16$. Construction times of a block tree includes parallel LPF array construction time.

construction time can be obtained by introducing stricter rules for the marking of nodes in the block tree rendering the pruning phase unnecessary. Finally, we want to compress the block tree recursively by using block trees internally.

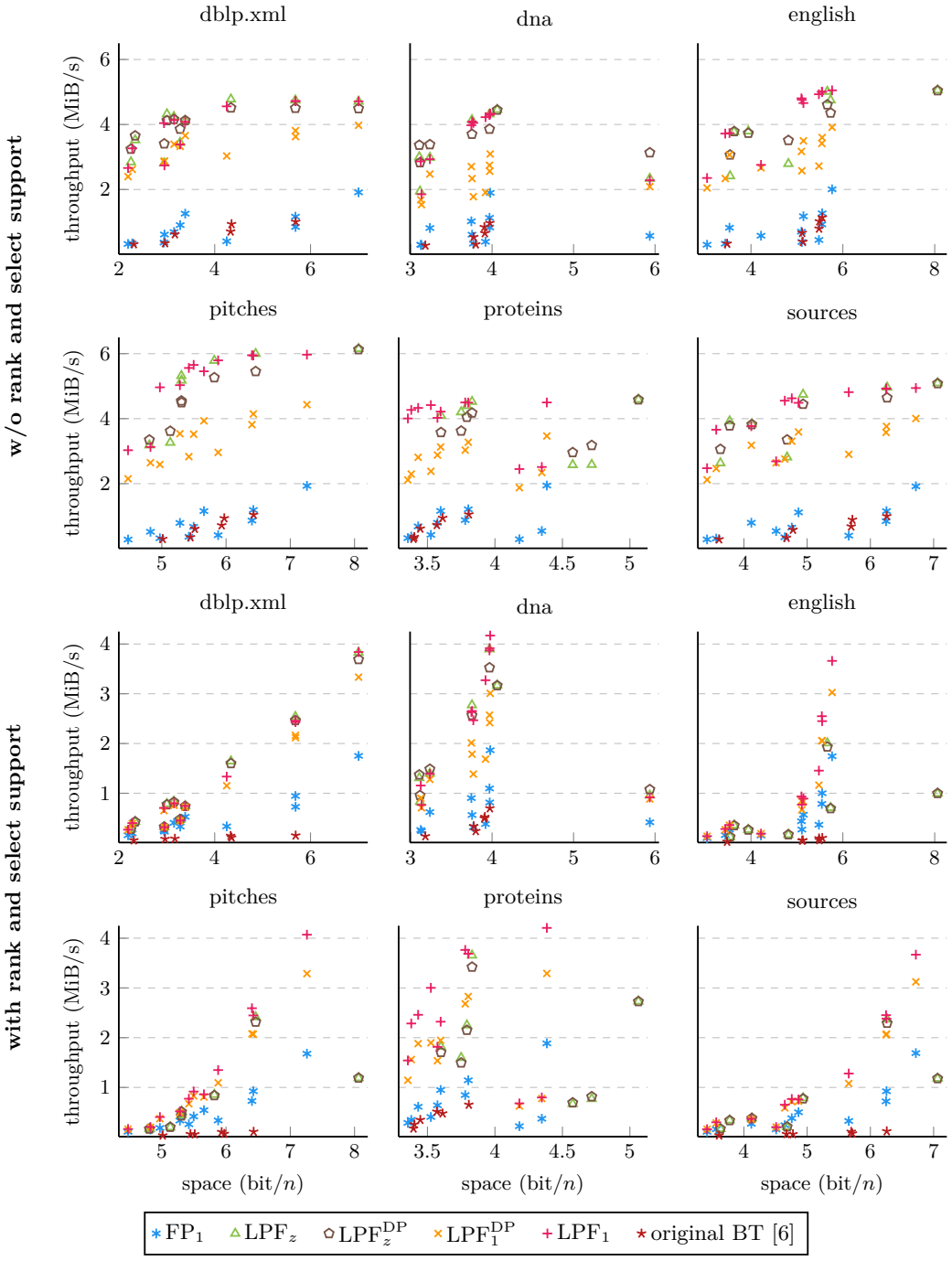



**Figure 5** Block tree construction throughput and space requirements per character of the input on 32 MiB prefixes of the non-repetitive inputs.

## References


1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.

2 Anisa Al-Hafeedh, Maxime Crochemore, Lucian Ilie, Evguenia Kopylova, William F. Smyth, German Tischler, and Munina Yusufu. A comparison of index-based Lempel-Ziv LZ77 factorization algorithms. *ACM Comput. Surv.*, 45(1):5:1–5:17, 2012. doi:10.1145/2379776.2379781.

3 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *CPM*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.22.

4 Jérémy Barbay, Francisco Claude, Travis Gagie, Gonzalo Navarro, and Yakov Nekrich. Efficient fully-compressed sequence representations. *Algorithmica*, 69(1):232–268, 2014. doi:10.1007/S00453-012-9726-3.

5 Jérémy Barbay, Meng He, J. Ian Munro, and Srinivasa Rao Satti. Succinct indexes for strings, binary relations and multilabeled trees. *ACM Trans. Algorithms*, 7(4):52:1–52:27, 2011. doi:10.1145/2000807.2000820.

6 Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *J. Comput. Syst. Sci.*, 117:1–22, 2021. doi:10.1016/j.jcss.2020.11.002.

7 Djamal Belazzougui, Patrick Hagge Cording, Simon J. Puglisi, and Yasuo Tabei. Access, rank, and select in grammar-compressed strings. In *ESA*, volume 9294 of *Lecture Notes in Computer Science*, pages 142–154. Springer, 2015. doi:10.1007/978-3-662-48350-3_13.

8 Djamal Belazzougui, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on lz-bounded encodings. In *DCC*, pages 83–92. IEEE, 2015. doi:10.1109/DCC.2015.69.

9 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015. doi:10.1145/2629339.

10 Antonio Boffa, Paolo Ferragina, and Giorgio Vinciguerra. A learned approach to design compressed rank/select data structures. *ACM Trans. Algorithms*, 18(3):24:1–24:28, 2022. doi:10.1145/3524060.

11 Nieves R. Brisaboa, Travis Gagie, Adrián Gómez-Brandón, and Gonzalo Navarro. Two-dimensional block trees. In *DCC*, pages 227–236. IEEE, 2018. doi:10.1109/DCC.2018.00031.

12 Manuel Cáceres and Gonzalo Navarro. Faster repetition-aware compressed suffix trees based on block trees. *Inf. Comput.*, 285(Part):104749, 2022. doi:10.1016/J.IC.2021.104749.

13 Matteo Ceregini, Florian Kurpicz, and Rossano Venturini. Faster wavelet trees with quad vectors. *CoRR*, abs/2302.09239, 2023. doi:10.48550/arXiv.2302.09239.

14 Yu-Feng Chien, Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Geometric BWT: compressed text indexing via sparse suffixes and range searching. *Algorithmica*, 71(2):258–278, 2015. doi:10.1007/S00453-013-9792-1.

15 Francisco Claude, Antonio Fariña, Miguel A. Martínez-Prieto, and Gonzalo Navarro. Universal indexes for highly repetitive document collections. *Inf. Syst.*, 61:1–23, 2016. doi:10.1016/J.IS.2016.04.002.

16 Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In *SPIRE*, volume 7608 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2012. doi:10.1007/978-3-642-34109-0_19.

17 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. doi:10.1016/J.IPL.2007.10.006.

18 Patrick Dinklage, Jonas Ellert, Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Practical wavelet tree construction. *ACM J. Exp. Algorithmics*, 26:1.8:1–1.8:67, 2021. doi:10.1145/3457197.

19 Patrick Dinklage, Johannes Fischer, and Florian Kurpicz. Constructing the wavelet tree and wavelet matrix in distributed memory. In *ALENEX*, pages 214–228. SIAM, 2020. doi:10.1137/1.9781611976007.17.

20 Patrick Dinklage, Johannes Fischer, Florian Kurpicz, and Jan-Philipp Tarnowski. Bit-parallel (compressed) wavelet tree construction. In *DCC*, pages 81–90. IEEE, 2023. doi:10.1109/DCC55655.2023.00016.

21 Jonas Ellert and Florian Kurpicz. Parallel external memory wavelet tree and wavelet matrix construction. In *SPIRE*, volume 11811 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2019. doi:10.1007/978-3-030-32686-9_28.

22 Paolo Ferragina, Raffaele Giancarlo, and Giovanni Manzini. The myriad virtues of wavelet trees. *Inf. Comput.*, 207(8):849–866, 2009. doi:10.1016/J.IC.2008.12.010.

23 Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. An alphabet-friendly fm-index. In *SPIRE*, volume 3246 of *Lecture Notes in Computer Science*, pages 150–160. Springer, 2004. doi:10.1007/978-3-540-30213-1_23.

24 Paolo Ferragina, Giovanni Manzini, and Giorgio Vinciguerra. Compressing and querying integer dictionaries under linearities and repetitions. *IEEE Access*, 10:118831–118848, 2022. doi:10.1109/ACCESS.2022.3221520.

25 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.*, 372(1):115–121, 2007. doi:10.1016/J.TCS.2006.12.012.

26 Johannes Fischer, Florian Kurpicz, and Marvin Löbel. Simple, fast and lightweight parallel wavelet tree construction. In *ALENEX*, pages 9–20. SIAM, 2018. doi:10.1137/1.9781611975055.2.

27 Frantisek Franek, Jan Holub, William F. Smyth, and Xiangdong Xiao. Computing quasi suffix arrays. *J. Autom. Lang. Comb.*, 8(4):593–606, 2003. doi:10.25596/JALC-2003-593.

28 José Fuentes-Sepúlveda, Erick Elejalde, Leo Ferres, and Diego Seco. Parallel construction of wavelet trees on multicore architectures. *Knowl. Inf. Syst.*, 51(3):1043–1066, 2017. doi:10.1007/s10115-016-1000-6.

29 Moses Ganardi, Artur Jez, and Markus Lohrey. Balancing straight-line programs. In *FOCS*, pages 1169–1183. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00073.

30 Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *SEA*, volume 8504 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 2014. doi:10.1007/978-3-319-07959-2_28.

31 Simon Gog and Matthias Petri. Optimized succinct data structures for massive data. *Softw. Pract. Exp.*, 44(11):1287–1314, 2014. doi:10.1002/SPE.2198.

32 Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *SODA*, pages 368–373. ACM Press, 2006.

33 Alexander Golynski, Rajeev Raman, and S. Srinivasa Rao. On the redundancy of succinct data structures. In *SWAT*, volume 5124 of *Lecture Notes in Computer Science*, pages 148–159. Springer, 2008. doi:10.1007/978-3-540-69903-3_15.

34 Rodrigo González, Szymon Grabowski, Veli Mäkinen, and Gonzalo Navarro. Practical implementation of rank and select queries. In *WEA*, pages 27–38, 2005.

35 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *SODA*, pages 841–850. ACM/SIAM, 2003.

36 Roberto Grossi, Alessio Orlandi, and Rajeev Raman. Optimal trade-offs for succinct string indexes. In *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 678–689. Springer, 2010. doi:10.1007/978-3-642-14165-2_57.

37 Roberto Grossi, Jeffrey Scott Vitter, and Bojian Xu. Wavelet trees: From theory to practice. In *CCP*, pages 210–221. IEEE Computer Society, 2011. doi:10.1109/CCP.2011.16.

38 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/RD.312.0249.

39 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *STOC*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

40 Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Towards a definitive measure of repetitiveness. In *LATIN*, volume 12118 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 2020. doi:10.1007/978-3-030-61792-9_17.

41 Dominik Köppl, Gonzalo Navarro, and Nicola Prezza. HOLZ: high-order entropy encoding of lempel-ziv factor distances. In *DCC*, pages 83–92. IEEE, 2022. doi:10.1109/DCC52660.2022.00016.

42 Florian Kurpicz. pasta::block_tree_experiments. doi:10.5281/zenodo.8114299.

43 Florian Kurpicz. Engineering compact data structures for rank and select queries on bit vectors. In *SPIRE*, volume 13617 of *Lecture Notes in Computer Science*, pages 257–272. Springer, 2022. doi:10.1007/978-3-031-20643-6_19.

44 Florian Kurpicz and Daniel Meyer. pasta::block_tree. doi:10.5281/zenodo.8114255.

45 Julian Labeit, Julian Shun, and Guy E. Blelloch. Parallel lightweight wavelet tree, suffix array and fm-index construction. *J. Discrete Algorithms*, 43:2–17, 2017. doi:10.1016/j.jda.2017.04.001.

46 Tobias Maier, Peter Sanders, and Roman Dementiev. Concurrent hash tables: Fast and general(?)! *ACM Trans. Parallel Comput.*, 5(4):16:1–16:32, 2019. doi:10.1145/3309206.

47 Stefano Marchini and Sebastiano Vigna. Compact fenwick trees for dynamic ranking and selection. *Softw. Pract. Exp.*, 50(7):1184–1202, 2020. doi:10.1002/spe.2791.

48 Daniel Meyer. Engineering block trees. Master's thesis, Karlsruhe Institute of Technology, 2022.

49 Gonzalo Navarro. A self-index on block trees. In *SPIRE*, volume 10508 of *Lecture Notes in Computer Science*, pages 278–289. Springer, 2017. doi:10.1007/978-3-319-67428-5_24.

50 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2022. doi:10.1145/3434399.

51 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2–es, 2007. doi:10.1145/1216370.1216372.

52 Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019. doi:10.1016/J.TCS.2018.09.007.

53 Gonzalo Navarro and Eliana Providel. Fast, small, simple rank/select on bitmaps. In *SEA*, volume 7276 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 2012. doi:10.1007/978-3-642-30850-5_26.

54 Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *ALENEX*. SIAM, 2007. doi:10.1137/1.9781611972870.6.

55 Mihai Pătraşcu. Succincter. In *FOCS*, pages 305–313. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.83.

56 Alberto Ordóñez Pereira, Gonzalo Navarro, and Nieves R. Brisaboa. Grammar compressed sequences with rank/select support. *J. Discrete Algorithms*, 43:54–71, 2017. doi:10.1016/J.JDA.2016.10.001.

57 Giulio Ermanno Pibiri and Shunsuke Kanda. Rank/select queries over mutable bitmaps. *Inf. Syst.*, 99:101756, 2021. doi:10.1016/j.is.2021.101756.

58 Nicola Prezza. Optimal rank and select queries on dictionary-compressed text. In *CPM*, volume 128 of *LIPIcs*, pages 4:1–4:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CPM.2019.4.

59 Nicola Prezza and Giovanna Rosone. Faster online computation of the succinct longest previous factor array. In *CiE*, volume 12098 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 2020. doi:10.1007/978-3-030-51466-2_31.

60 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding $k$-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43, 2007. doi:10.1145/1290672.1290680.

61 Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam D. Smith. Sublinear algorithms for approximating string compressibility. *Algorithmica*, 65(3):685–709, 2013. doi:10.1007/s00453-012-9618-6.

62 Julian Shun and Fuyao Zhao. Practical parallel Lempel-Ziv factorization. In *DCC*, pages 123–132. IEEE, 2013. doi:10.1109/DCC.2013.20.

63 Zachary Stephens, Skylar Lee, Faraz Faghri, Roy Campbell, Chengxiang Zhai, Miles Efron, Ravishankar Iyer, Michael Schatz, Saurabh Sinha, and Gene Robinson. Big data: Astronomical or genomical? *PLoS biology*, 13(7):1–11, 2015.

64 Sebastiano Vigna. Broadword implementation of rank/select queries. In *WEA*, volume 5038 of *Lecture Notes in Computer Science*, pages 154–168. Springer, 2008. doi:10.1007/978-3-540-68552-4_12.

65 Dong Zhou, David G. Andersen, and Michael Kaminsky. Space-efficient, high-performance rank and select structures on uncompressed bit sequences. In *SEA*, volume 7933 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 2013. doi:10.1007/978-3-642-38527-8_15.

66 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

# Connectivity Queries Under Vertex Failures: Not Optimal, but Practical

**Evangelos Kosinas**
Department of Computer Science & Engineering, University of Ioannina, Greece

## Abstract

We revisit once more the problem of designing an oracle for answering connectivity queries in undirected graphs in the presence of vertex failures. Specifically, given an undirected graph $G$ with $n$ vertices and $m$ edges and an integer $d_\star \ll n$, the goal is to preprocess the graph in order to construct a data structure $\mathcal{D}$ such that, given a set of vertices $F$ with $|F| = d \leq d_\star$, we can derive an oracle from $\mathcal{D}$ that can efficiently answer queries of the form "is $x$ connected with $y$ in $G \setminus F$?". Very recently, Long and Saranurak (FOCS 2022) provided a solution to this problem that is almost optimal with respect to the preprocessing time, the space usage, the update time, and the query time. However, their solution is highly complicated, and it seems very difficult to be implemented efficiently. Furthermore, it does not settle the complexity of the problem in the regime where $d_\star$ is a constant. Here, we provide a much simpler solution to this problem, that uses only textbook data structures. Our algorithm is deterministic, it has preprocessing time and space complexity $O(d_\star m \log n)$, update time $O(d^4 \log n)$, and query time $O(d)$. These bounds compare very well with the previous best, especially considering the simplicity of our approach. In fact, if we assume that $d_\star$ is a constant ($d_\star \geq 4$), then our algorithm provides some trade-offs that improve the state of the art in some respects. Finally, the data structure that we provide is flexible with respect to $d_\star$: it can be adapted to increases and decreases, in time and space that are almost proportional to the change in $d_\star$ and the size of the graph.

**2012 ACM Subject Classification** Mathematics of computing → Paths and connectivity problems; Theory of computation → Graph algorithms analysis

**Keywords and phrases** Graphs, Connectivity, Fault-Tolerant, Oracles

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.75

**Related Version** *Full Version*: https://arxiv.org/abs/2305.01756

**Funding** The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 3rd Call for HFRI PhD Fellowships (Fellowship Number: 6547.).

**Acknowledgements** I want to thank my advisor, Loukas Georgiadis, for helpful comments on this manuscript. I also want to thank the anonymous reviewers for their useful suggestions.

## 1 Introduction

In this paper we deal with the following problem. Given an undirected graph $G$ with $n$ vertices and $m$ edges, and a fixed integer $d_\star$ ($d_\star \ll n$), the goal is to construct a data structure $\mathcal{D}$ that can be used in order to answer connectivity queries in the presence of at most $d_\star$ vertex-failures. More precisely, given a set of vertices $F$, with $|F| \leq d_\star$, we must be able to efficiently derive an oracle from $\mathcal{D}$, which can efficiently answer queries of the form "are the vertices $x$ and $y$ connected in $G \setminus F$?". In this problem, we want to simultaneously optimize the following parameters: (1) the construction time of $\mathcal{D}$ (preprocessing time), (2) the space usage of $\mathcal{D}$, (3) the time to derive the oracle from $\mathcal{D}$ given $F$ (update time), and (4) the time to answer a connectivity query in $G \setminus F$. This problem is very well motivated; it has attracted the attention of researchers for more than a decade now, and it has many interesting variations. The reader is referred to [5] or [7] for the details on its history and its variations.


## 1.1 Previous work

Despite being extensively studied, it is only very recently that an almost optimal solution was provided by Long and Saranurak [7]. Specifically, they provided a deterministic algorithm that has $\hat{O}(m) + \tilde{O}(d_\star m)$ preprocessing time, uses $O(m \log^* n)$ space, and has $\hat{O}(d^2)$ update time and $O(d)$ query time.[1] This improves on the previous best deterministic solution by Duan and Pettie [5], that has $O(mn \log n)$ preprocessing time, uses $O(d_\star m \log n)$ space, and has $O(d^3 \log^3 n)$ update time and $O(d)$ query time. We note that there are more solutions to this problem, that optimize some parameters while sacrifising others (e.g., in the solution of Pilipczuk et al. [9], there is no dependency on $n$ in the update time, but this is superexponential in $d_\star$, and the preprocessing time is $O(mn^2 2^{2^{O(d_\star)}})$). We refer to Table 1 in reference [7] for more details on the best known (upper) bounds for this problem. We also refer to Theorem 1.2 in [7] for a summary of known (conditional) lower bounds, that establish the optimality of [7].

## 1.2 Our contribution

The bounds that we mentioned are the best known for a deterministic solution. In practice, one would prefer the solution of Long and Saranurak [7], because that of Duan and Pettie [5] has preprocessing time $O(mn \log n)$, which can be prohibitively slow for large enough graphs. However, the solution in [7] is highly complicated, and it seems very difficult to be implemented efficiently. This is a huge gap between theory and practice. Furthermore, the (hidden) dependence on $n$ in the time-bounds of [7] is not necessarily optimal if we assume that $d_\star$ is a constant for our problem. We note that this is a problem with various parameters, and thus it is very difficult to optimize all of them simultaneously.

Considering that this is a fundamental connectivity problem, we believe that it is important to have a solution that is relatively simple to describe and analyze, compares very well with the best known bounds (even improves them in some respects), opens a new direction to settle the complexity of the problem, and can be readily implemented efficiently.

In this paper, we exhibit a solution that has precisely those characteristics. We present a deterministic algorithm that has preprocessing time $O(d_\star m \log n)$, uses space $O(d_\star m \log n)$, and has $O(d^4 \log n)$ update time and $O(d)$ query time.[2] Our approach is arguably the simplest that has been proposed for this problem. The previous solutions rely on sophisticated tree decompositions of the original graph. Here, instead, we basically rely on a single DFS-tree, and we simply analyze its connected components after the removal of a set of vertices. It turns out that there is enough structure to allow for an efficient solution (see Section 3.2).

The bounds that we provide compare very well with the previous best, especially considering the simplicity of our approach. (See Tables 1 and 2.) In fact, as we can see in Table 1, our solution is the best choice for implementations, considering that the algorithm of Long and Saranurak is very difficult to be implemented within the claimed time-bounds. Furthermore, if we assume that $d_\star$ is a constant ($d_\star \geq 4$), then, as we can see in Table 2, our algorithm provides some trade-offs, that improve the state of the art in some respects.

---

[1] The symbol $\hat{O}$ hides subpolynomial (i.e. $n^{o(1)}$) factors, and $\tilde{O}$ hides polylogarithmic factors. The hidden expressions in the time-bounds are not specified by the authors in their overview. Also, the description for the $\log^* n$ function that appears in the space complexity is that it "can be substituted with any slowly growing function". One thing that is explicitly stated, however, is that the hidden subpolynomial factors are worse than polylogarithmic. We must emphasize that the difficulty in stating the precise bounds is partly due to there being various trade-offs in the functions involved, and is partly indicative of the complexity of the techniques that are used.

[2] The log factors in the space usage and the time for the updates can be improved with the use of more sophisticated 2D-range-emptiness data structures, such as those in [2].

**Table 1** Comparison of the best-known deterministic bounds. We note that $m$ can be replaced with $\bar{m} = min\{m, d_\star n\}$, using the sparsification of Nagamochi and Ibaraki [8]. The data structure of Pilipczuk et al. does not support an update phase, but answers queries directly, given a set of (at most $d_\star$) failed vertices and two query vertices.

| | Preprocessing | Space | Update | Query |
|---|---|---|---|---|
| Pilipczuk et al. [9] | $O(2^{2^{O(d_\star)}} mn^2)$ | $O(2^{2^{O(d_\star)}} m)$ | – | $O(2^{2^{O(d_\star)}})$ |
| Duan and Pettie [5] | $O(mn \log n)$ | $O(d_\star m \log n)$ | $O(d^3 \log^3 n)$ | $O(d)$ |
| Long and Saranurak [7] | $\hat{O}(m) + \tilde{O}(d_\star m)$ | $O(m \log^* n)$ | $\hat{O}(d^2)$ | $O(d)$ |
| **This paper** | $O(d_\star m \log n)$ | $O(d_\star m \log n)$ | $O(d^4 \log n)$ | $O(d)$ |

**Table 2** Comparison of the best-known deterministic bounds, when $d_\star$ is a fixed (small) constant. Although the algorithm of Pilipczuk et al. has the best space and query-time bounds, it has very large preprocessing time. Our solution has the best preprocessing time, and also better update time compared to the solutions of [5] and [7]. Furthermore, our space usage is almost linear.

| | Preprocessing | Space | Update | Query |
|---|---|---|---|---|
| Pilipczuk et al. [9] | $O(mn^2)$ | $O(m)$ | – | $O(1)$ |
| Duan and Pettie [5] | $O(mn \log n)$ | $O(m \log n)$ | $O(\log^3 n)$ | $O(1)$ |
| Long and Saranurak [7] | $\hat{O}(m) + \tilde{O}(m)$ | $O(m \log^* n)$ | $\hat{O}(1)$ | $O(1)$ |
| **This paper** | $O(m \log n)$ | $O(m \log n)$ | $O(\log n)$ | $O(1)$ |

Finally, the data structure that we provide is flexible with respect to $d_\star$: it can be adapted to increases and decreases, in time and space that are almost proportional to the change in $d_\star$ and the size of the graph (see Corollary 3). We do not know if any of the previous solutions has this property. It is a natural question whether we can efficiently update the data structure so that it can handle more failures (or less, and thereby free some space). As far as we know, we are the first to take notice of this aspect of the problem.

## 2 Preliminaries

We assume that the reader is familiar with standard graph-theoretical terminology (see, e.g., [4]). The notation that we use is also standard. Since we deal with connectivity under *vertex* failures, it is sufficient to consider simple graphs as input to our problem (because the existence of parallel edges does not affect the connectivity relation). However, during the update phase, we construct a multigraph that represents the connectivity relationship between some connected components after removing the failed vertices (Definition 9). The parallel edges in this graph are redundant, but they may be introduced by the algorithm that we use to construct it, and it would be costly to check for redundancy throughout.

It is also sufficient to assume that the input graph $G$ is connected. Because, otherwise, we can initialize a data structure on every connected component of $G$; the updates, for a given set of failures, are distributed to the data structures on the connected components, and the queries for pairs of vertices that lie in different connected components of $G$ are always *false*. We use $G$ to denote the input graph throughout; $n$ and $m$ denote its number of vertices and edges, respectively. For any two integers $x, y$, we use the interval notation $[x, y]$ to denote the set $\{x, x+1, \ldots, y\}$. (If $x > y$, then $[x, y] = \emptyset$.)

## 2.1 DFS-based concepts

Let $T$ be a DFS-tree of $G$, with start vertex $r$ [10]. We use $p(v)$ to denote the parent of every vertex $v \neq r$ in $T$ ($v$ is a child of $p(v)$). For any two vertices $u, v$, we let $T[u, v]$ denote the simple tree path from $u$ to $v$ on $T$. For every two vertices $u$ and $v$, if the tree path $T[r, u]$ uses $v$, then we say that $v$ is an ancestor of $u$ (equivalently, $u$ is a descendant of $v$). In particular, a vertex is considered to be an ancestor (and also a descendant) of itself. It is very useful to identify the vertices with their order of visit during the DFS, starting with $r \leftarrow 1$. Thus, if $v$ is an ancestor of $u$, we have $v < u$. For any vertex $v$, we let $T(v)$ denote the subtree rooted at $v$, and we let $ND(v)$ denote the number of descendants of $v$ (i.e., $ND(v) = |T(v)|$). Thus, we have that $T(v) = [v, v + ND(v) - 1]$, and therefore we can check the ancestry relation in constant time. Two children $c$ and $c'$ of a vertex $v$ are called *consecutive children* of $v$ (in this order), if $c'$ is the minimum child of $v$ with $c' > c$. Notice that, in this case, we have $T(c) \cup T(c') = [c, c' + ND(c') - 1]$.

A DFS-tree has the following extremely convenient property: the endpoints of every non-tree edge of $G$ are related as ancestor and descendant [10], and so we call those edges *back-edges*. Our whole approach is basically an exploitation of this property, which does not hold in general rooted spanning trees of $G$ (unless they are derived from a DFS traversal, and only then [10]). To see why this is relevant for our purposes, consider what happens when we remove a vertex $f \neq r$ from $T$. Let $c_1, \ldots, c_k$ be the children of $f$ in $T$. Then, the connected components of $T \setminus f$ are given by $T(c_1), \ldots, T(c_k)$ and $T(r) \setminus T(f)$. A subtree $T(c_i)$, $i \in \{1, \ldots, k\}$, is connected with the rest of the graph in $G \setminus f$ if and only if there is a back-edge that stems from $T(c_i)$ and ends in a proper ancestor of $f$. Now, this problem has an algorithmically elegant solution. Suppose that we have computed, for every vertex $v \neq r$, the *lowest* proper ancestor of $v$ that is connected with $T(v)$ through a back-edge. We denote this vertex as $low(v)$. Then, we may simply check whether $low(c_i) < f$, in order to determine whether $T(c_i)$ is connected with $T(r) \setminus T(f)$ in $G \setminus f$.

We extend the concept of the $low$ points, by introducing the $low_k$ points, for any $k \in \mathbb{N}$. These are defined recursively, for any vertex $v \neq r$, as follows. $low_1(v)$ coincides with $low(v)$. Then, supposing that we have defined $low_k(v)$ for some $k \in \mathbb{N}$, we define $low_{k+1}(v)$ as $min(\{y \mid \exists \text{ a back-edge } (x, y) \text{ such that } x \in T(v) \text{ and } y < v\} \setminus \{low_1(v), \ldots, low_k(v)\})$. Notice that $low_k(v)$ may not exist for some $k \in \mathbb{N}$ (and this implies that $low_{k'}(v)$ does not exist, for any $k' > k$). If, however, $low_k(v)$ exists, then $low_{k'}(v)$, for any $k' < k$, also exists, and we have $low_1(v) < low_2(v) < \cdots < low_k(v)$. Notice that the existence of $low_k(v)$ implies that there is a back-edge $(x, low_k(v))$, where $x$ is a descendant of $v$.

▶ **Proposition 1** ([6]). *Let $T$ be a DFS-tree of a simple graph $G$, and assume that the adjacency list of every vertex of $G$ is sorted in increasing order w.r.t. the DFS numbering. Suppose also that, for some $k \in \{0, \ldots, n-1\}$, we have computed the $low_1, \ldots, low_k$ points of all vertices (w.r.t. $T$), and the set $\{low_1(v), \ldots, low_k(v)\}$ is stored in an increasingly sorted array for every $v \neq r$. Then we can compute the $low_{k+1}$ points of all vertices in $O(n \log(k+1))$ time.*[3]

▶ **Corollary 2** ([6]). *For any $k \in \{1, \ldots, n-1\}$, the $low_1, \ldots, low_k$ points of all vertices can be computed in $O(m + kn \log k)$ time.*

[3] We make the convention that $\log(1) = 1$, so that the time to compute the $low_1$ points is $O(n)$.

# 3 The algorithm for vertex failures

## 3.1 Initializing the data structure

We will need the following ingredients in order to be able to handle at most $d_\star$ failed vertices.

- **(i)** A DFS-tree $T$ of $G$ rooted at a vertex $r$. The values $ND$ and $depth$ (w.r.t. $T$) must be computed for all vertices. We identify the vertices of $G$ with the DFS numbering of $T$.
- **(ii)** A level-ancestor data structure on $T$.
- **(iii)** A 2D-range-emptiness data structure on the set of the back-edges of $G$ w.r.t. $T$.
- **(iv)** The $low_i$ points of all vertices, for every $i \in \{1, \ldots, d_\star\}$.
- **(v)** For every $i \in \{1, \ldots, d_\star\}$, a DFS-tree $T_i$ of $T$ rooted at $r$, where the adjacency lists of the vertices are given by their children lists sorted in increasing order w.r.t. the $low_i$ point.
- **(vi)** For every $i \in \{1, \ldots, d_\star\}$, a 2D-range-emptiness data structure on the set of the back-edges of $G$ w.r.t. $T_i$.

The $depth$ value in $(i)$ refers to the depths of the vertices in $T$. This is defined for every vertex $v$ as the size of the tree path $T[r, v]$. (Thus, e.g., $depth(r) = 1$.) It takes $O(n)$ additional time to compute the $depth$ values during the DFS.

The level-ancestor data structure in $(ii)$ is used in order to answer queries of the form $\texttt{QueryLA}(v, \delta) \equiv$ "return the ancestor of $v$ that lies at depth $\delta$". We use those queries in order to find the children of vertices that are ancestors of other vertices. (I.e., given that $u$ is a descendant of $v$, we want to know the child of $v$ that is an ancestor of $u$.) For our purposes, it is sufficient to use the solution in Section 3 of [1], that preprocesses $T$ in $O(n \log n)$ time so that it can answer level-ancestor queries in (worst-case) $O(1)$ time.

The 2D-range-emptiness data structure in $(iii)$ is used in order to answer queries of the form $\texttt{2D\_range}([X_1, X_2] \times [Y_1, Y_2]) \equiv$ "is there a back-edge $(x, y)$ with $x \in [X_1, X_2]$ and $y \in [Y_1, Y_2]$?".[4] We can use a standard implementation for this data structure, that has $O(m \log n)$ space and preprocessing time complexity, and can answer a query in (worst-case) $O(\log n)$ time (see, e.g., Section 5.6 in [3]). The $m$ factor here is unavoidable, because the number of back-edges can be as large as $m - n + 1$. However, we note that we can improve the $\log n$ factor in the space and the query time if we use a more sophisticated solution, such as [2].

The $low_1, \ldots, low_{d_\star}$ points of all vertices can be computed in $O(m + d_\star n \log d_\star) = O(m + d_\star n \log n)$ time (Corollary 2). We obviously need $O(d_\star n)$ space to store them.

For $(v)$, we just perform $d_\star$ DFS's on $T$, starting from $r$, where each time we use a different arrangement of the children lists of $T$ as adjacency lists. This takes $O(d_\star n)$ time in total, but we do not need to actually store the trees. (In fact, the parent pointer is the same for all of them.) What we actually need here is the DFS numbering of the $i$-th DFS traversal, for every $i \in \{1, \ldots, d_\star\}$, which we denote as $DFS_i$. We keep those DFS numberings stored, and so we need $O(d_\star n)$ additional space. The usefulness of performing all those DFS's will become clear in Section 3.4. Right now, we only need to mention that, for every $i \in \{1, \ldots, d_\star\}$, the ancestry relation in $T_i$ is the same as that in $T$. Thus, the $low_1, \ldots, low_{d_\star}$ points for all vertices w.r.t. $T_i$ are the same as those w.r.t. $T$.

The 2D-range-emptiness data structures in $(vi)$ are used in order to answer queries of the form $\texttt{2D\_range\_i}([X_1, X_2] \times [Y_1, Y_2]) \equiv$ "is there a back-edge $(x, y)$ with $x \in [X_1, X_2]$ and $y \in [Y_1, Y_2]$?", where the endpoints of the query rectangle refer to the $DFS_i$ numbering,

[4] The input to `2D_range` is just the endpoints $X_1, X_2, Y_1, Y_2$ of the query rectangle; we use brackets around them, and the symbol $\times$, just for readability.

for $i \in \{1, \ldots, d_\star\}$. Since the ancestry relation is the same for $T_i$ and $T$, we have that the queries `2D_range`$([X_1, X_2] \times [Y_1, Y_2])$ and `2D_range_i`$([X_1, X_2]_i \times [Y_1, Y_2]_i)$ are equivalent, where the $i$ index below the brackets means that we have translated the endpoints in the $DFS_i$ numbering.

The construction of the 2D-range-emptiness data structures w.r.t. the DFS-trees $T_1, \ldots, T_{d_\star}$ takes $O(d_\star m \log n)$ time in total. In order to keed those data structures stored, we need $O(d_\star m \log n)$ space. Thus, the construction and the storage of the 2D-range-emptiness data structures dominate the space-time complexity overall.

It is easy to see that the list of data structures from $(i)$ to $(vi)$ is flexible w.r.t. $d_\star$. Thus, if $d_\star$ increases by 1, then we need to additionally compute the $low_{d_\star+1}$ points of all vertices, the $T_{d_\star+1}$ DFS-tree, and the corresponding 2D-range-emptiness data structure. Computing the $low_{d_\star+1}$ points takes $O(n \log(d_\star + 1)) = O(n \log n)$ time, and demands an additional $O(n)$ space, assuming that we have sorted the adjacency lists of $G$ in increasing order, and that we have stored the $low_1, \ldots, low_{d_\star}$ points, for every vertex, in an increasingly sorted array (see Proposition 1).

▶ **Corollary 3.** *Suppose that we have initialized our data structure for some $d_\star$, and we want to get a data structure for $d_\star + k$. Then we can achieve this in $O(km \log n)$ time, using extra $O(km \log n)$ space.*

If $d_\star$ decreases by $k$, then we just have to discard the $low_{d_\star-k+1}, \ldots, low_{d_\star}$ points, the $T_{d_\star-k+1}, \ldots, T_{d_\star}$ DFS-trees, and the corresponding 2D-range-emptiness data structures. This will free $O(km \log n)$ space.

## 3.2 The general idea

Let $F$ be a set of failed vertices. Then $T \setminus F$ may consist of several connected components, all of which are subtrees of $T$. It will be necessary to distinguish two types of connected components of $T \setminus F$. Let $C$ be a connected component of $T \setminus F$. If no vertex in $F$ is a descendant of $C$, then $C$ is called a *hanging subtree* of $T \setminus F$. Otherwise, $C$ is called an *internal component* of $T \setminus F$. (See Figure 1 for an illustration.) Observe that, while the number of connected components of $T \setminus F$ may be as large as $n - 1$ (even if $|F| = 1$), the number of internal components of $T \setminus F$ is at most $|F|$. This is an important observation, that allows us to reduce the connectivity of $G \setminus F$ to the connectivity of the internal components.

More precisely, we can already provide a high level description of our strategy for answering connectivity queries between pairs of vertices. Let $x, y$ be two vertices of $G \setminus F$. Suppose first that $x$ belongs to an internal component $C_1$ and $y$ belongs to an internal component $C_2$. Then it is sufficient to know whether $C_1$ and $C_2$ are connected in $G \setminus F$. Otherwise, if either $x$ or $y$ lies in a hanging subtree $C$, then we can substitute $C$ with any internal component that is connected with $C$ in $G \setminus F$. If no such internal component exists, then $x$ and $y$ are connected in $G \setminus F$ if and only if they lie in the same hanging subtree.

Thus, after the deletion of $F$ from $G$, it is sufficient to make provisions so as to be able to efficiently answer the following:

**(1)** Given a vertex $x$, determine the connected component of $T \setminus F$ that contains $x$.

**(2)** Given two internal components $C_1$ and $C_2$ of $T \setminus F$, determine whether $C_1$ and $C_2$ are connected in $G \setminus F$.

**(3)** Given a hanging subtree $C$ of $T \setminus F$, find an internal component of $T \setminus F$ that is connected with $C$ in $G \setminus F$, or report that no such internal component exists.

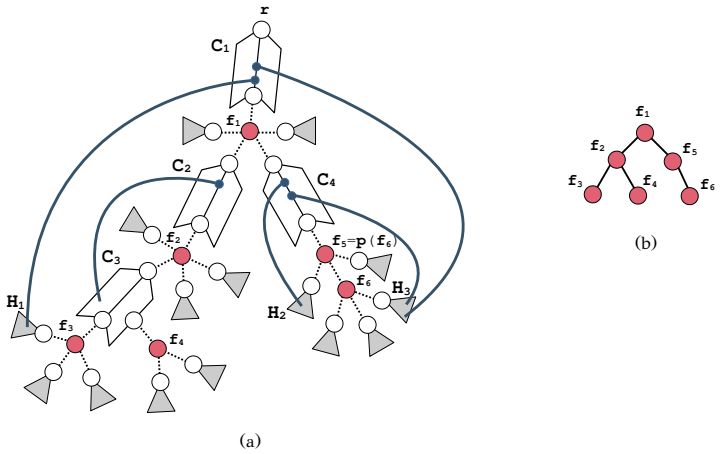



**Figure 1** (a) A set of failed vertices $F = \{f_1, \ldots, f_6\}$ on a DFS-tree $T$, and (b) the corresponding F-forest, which shows the $parent_F$ relation between failed vertices. Notice that $T \setminus F$ is split into several connected components, but there are only four internal components, $C_1$, $C_2$, $C_3$ and $C_4$. The hanging subtrees of $T \setminus F$ are shown with gray color (e.g., $H_1$, $H_2$ and $H_3$). The internal components $C_2$ and $C_3$ remain connected in $G \setminus F$ through a back-edge that connects them directly. $C_1$ and $C_4$ remain connected through the hanging subtree $H_3$ of $f_6$. We have $\partial(C_1) = \{f_1\}$, $\partial(C_2) = \{f_2\}$, $\partial(C_3) = \{f_3, f_4\}$ and $\partial(C_4) = \{f_5\}$. Notice that $f_6$ is the only failed vertex that is not a boundary vertex of an internal component, and it has $parent_F(f_6) = p(f_6)$.

Actually, the most difficult task, and the only one that we provide a preprocessing for (during the update phase), is (2). We explain how to perform (1) and (3) during the process of answering a query, in Section 3.5. An efficient solution for (2) is provided in Section 3.4.

The general idea is that, since there are at most $d = |F|$ internal components of $T \setminus F$, we can construct a graph with $O(d)$ nodes, representing the internal components of $T \setminus F$, that captures the connectivity relation among them in $G \setminus F$ (see Lemma 10). This is basically done with the introduction of some artificial edges between the (representatives of the) internal components. In the following subsection, we state some lemmas concerning the structure of the internal components, and their connectivity relationship in $G \setminus F$. All omitted proofs are contained in the full version of our paper [6].

## 3.3 The structure of the internal components

We will use the roots of the connected components of $T \setminus F$ (viewed as rooted subtrees of $T$) as representantives of them. Now we introduce some terminology and notation. If $C$ is a connected component of $T \setminus F$, we denote its root as $r_C$. If $C$ is a hanging subtree of $T \setminus F$, then $p(r_C) = f$ is a failed vertex, and we say that $C$ is *a hanging subtree of* $f$. If $C, C'$ are two distinct connected components of $T \setminus F$ such that $r_{C'}$ is an ancestor of $r_C$, then we say that $C'$ is an ancestor of $C$. Furthermore, if $v$ is a vertex not in $C$ such that $v$ is an ancestor (resp., a descendant) of $r_C$, then we say that $v$ is an ancestor (resp., a descendant) of $C$. If $C$ is an internal component of $T \setminus F$ and $f$ is a failed vertex such that $p(f) \in C$, then we say that $f$ is a boundary vertex of $C$. The collection of all boundary vertices of $C$ is denoted as $\partial(C)$. Notice that any vertex $b \in \partial(C)$ has the property that there is no failed vertex on the tree path $T[p(b), r_C]$. Conversely, a failed vertex $b$ such that there is no failed vertex on the tree path $T[p(b), r_C]$ is a boundary vertex of $C$. Thus, if $b_1, \ldots, b_k$ is the collection of the boundary vertices of $C$, then $C = T(r_C) \setminus (T(b_1) \cup \cdots \cup T(b_k))$.

The following lemma is a set of properties that are satisfied by the internal components.

▶ **Lemma 4** ([6]). *Let $C$ be an internal component of $T \setminus F$. Then:*

**(1)** *Either $r_C = r$, or $p(r_C) \in F$.*

**(2)** *For every vertex $v$ that is a descendant of $C$, there is a unique boundary vertex of $C$ that is an ancestor of $v$.*

**(3)** *Let $f_1, \ldots, f_k$ be the boundary vertices of $C$, sorted in increasing order. Then $C$ is the union of the following subsets of consecutive vertices: $[r_C, f_1 - 1], [f_1 + ND(f_1), f_2 - 1], \ldots, [f_{k-1} + ND(f_{k-1}), f_k - 1], [f_k + ND(f_k), r_C + ND(r_C) - 1]$. (We note that some of those sets may be empty.)*

We represent the ancestry relation between failed vertices using a forest which we call the *failed vertex forest* (*F-forest*, for short). The F-forest consists of the following two elements. First, for every failed vertex $f$, there is a pointer $parent_F(f)$ to the nearest ancestor of $f$ (in $T$) that is also a failed vertex. If there is no ancestor of $f$ that is a failed vertex, then we let $parent_F(f) = \perp$. And second, every failed vertex $f$ has a pointer to its list of children in the F-forest.

The F-forest can be easily constructed in $O(d^2)$ time: we just have to find, for every failed vertex $f$, the maximum failed vertex $f'$ that is a proper ancestor of $f$; then we set $parent_F(f) = f'$, and we append $f$ to the list of the children of $f'$ in the F-forest.

The next lemma shows how we can check in constant time whether a failed vertex belongs to the boundary of an internal component, and how to retrieve the root of this component.

▶ **Lemma 5** ([6]). *A failed vertex $f$ is a boundary vertex of an internal component if and only if $parent_F(f) \neq p(f)$. Now let $f$ be a boundary vertex of an internal component $C$. Then, if $parent_F(f)$ exists, we have that the root of $C$ is the child of $parent_F(f)$ that is an ancestor of $f$. Otherwise, the root of $C$ is $r$.*

Thus, according to Lemma 5, if $f$ is a boundary vertex of an internal component $C$ with $r_C \neq r$, we can retrieve $r_C$ in constant time using a level-ancestor query: i.e., we ask for the ancestor of $f$ (in $T$) whose depth equals that of $parent_F(f) + 1$. We may use this fact throughout without mention.

The following lemma shows that there are two types of edges that determine the connectivity relation in $G \setminus F$ between the connected components of $T \setminus F$.

▶ **Lemma 6** ([6]). *Let $e$ be an edge of $G \setminus F$ whose endpoints lie in different connected components of $T \setminus F$. Then $e$ is a back-edge and either $(i)$ both endpoints of $e$ lie in internal components, or $(ii)$ one endpoint of $e$ lies in a hanging subtree $H$, and the other endpoint lies in an internal component $C$ that is an ancestor of $H$.*

▶ **Corollary 7** ([6]). *Let $C, C'$ be two distinct connected components of $T \setminus F$ that are connected with an edge $e$ of $G \setminus F$. Assume w.l.o.g. that $r_{C'} < r_C$. Then $C'$ is an ancestor of $C$.*

The following lemma provides an algorithmically useful criterion to determine whether a connected component of $T \setminus F$ – a hanging subtree or an internal component – is connected with an internal component of $T \setminus F$ through a back-edge.

▶ **Lemma 8** ([6]). *Let $C, C'$ be two connected components of $T \setminus F$ such that $C'$ is an internal component that is an ancestor of $C$, and let $b$ be the boundary vertex of $C'$ that is an ancestor of $C$. Then there is a back-edge from $C$ to $C'$ if and only if there is a back-edge from $C$ whose lower end lies in $[r_{C'}, p(b)]$.*

▶ **Definition 9** (Connectivity graph). *Let $\mathcal{R}$ be a multigraph where $V(\mathcal{R})$ is the set of the roots of the internal components of $T \setminus F$, and $E(\mathcal{R})$ satisfies the following three properties:*

**(1)** *For every back-edge connecting two internal components $C$ and $C'$, there is an edge $(r_C, r_{C'})$ in $\mathcal{R}$.*
**(2)** *Let $H$ be a hanging subtree of a failed vertex $f$, and let $C_1, \ldots, C_k$ be the internal components that are connected with $H$ through a back-edge. (By Lemma 6, all of $C_1, \ldots, C_k$ are ancestors of $H$.) Assume w.l.o.g. that $C_k$ is an ancestor of all $C_1, \ldots, C_{k-1}$. Then $\mathcal{R}$ contains the edges $(r_{C_1}, r_{C_k}), (r_{C_2}, r_{C_k}), \ldots, (r_{C_{k-1}}, r_{C_k})$.*
**(3)** *Every edge of $\mathcal{R}$ is given by either (1) or (2), or it is an edge of the form $(r_C, r_{C'})$, where $C, C'$ are two internal components that are connected in $G \setminus F$.*

*Then $\mathcal{R}$ is called a* connectivity graph of the internal components *of $T \setminus F$. The edges of (1) and (2) are called Type-1 and Type-2, respectively.*

The following lemma shows that a connectivity graph captures the connectivity relationship of the internal components of $T \setminus F$ in $G \setminus F$.

▶ **Lemma 10** ([6]). *Let $\mathcal{R}$ be a connectivity graph of the internal components of $T \setminus F$. Then, two internal components $C, C'$ of $T \setminus F$ are connected in $G \setminus F$ if and only if $r_C, r_{C'}$ are connected in $\mathcal{R}$.*

## 3.4 Handling the updates: construction of a connectivity graph for the internal components of $T \setminus F$

Given a set of failed vertices $F$, with $|F| = d \leq d_\star$, we will show how we can construct a connectivity graph $\mathcal{R}$ for the internal components of $T \setminus F$, using $O(d^4)$ calls to 2D-range-emptiness queries. Recall that $V(\mathcal{R})$ is the set of the roots of the internal components of $T \setminus F$.

Algorithm 1 shows how we can find all Type-1 edges of $\mathcal{R}$. The idea is basically to perform 2D-range-emptiness queries for every pair of internal components, in order to determine the existence of a back-edge that connects them. More precisely, we work as follows. Let $C$ be an internal component of $T \setminus F$. Then it is sufficient to check every ancestor component $C'$ of $C$, in order to determine whether there is a back-edge from $C$ to $C'$ (see Corollary 7). Let $f_1, \ldots, f_k$ be the boundary vertices of $C$, sorted in increasing order. Let also $f'$ be the boundary vertex of $C'$ that is an ancestor of $C$, and let $I = [r_{C'}, p(f')]$. Then we perform 2D-range-emptiness queries for the existence of a back-edge on the rectangles $[r_C, f_1 - 1] \times I, [f_1 + ND(f_1), f_2 - 1] \times I, \ldots, [f_k + ND(f_k), r_C + ND(r_C) - 1] \times I$. We know that there is a back-edge connecting $C$ and $C'$ if and only if at least one of those queries is positive (see Lemma 4(3) and Lemma 8). If that is the case, then we add the edge $(r_C, r_{C'})$ to $\mathcal{R}$.

Observe that the total number of 2D-range-emptiness queries that we perform is $O(d^2)$, because every one of them corresponds to a triple $(C, f, C')$, where $C, C'$ are internal components, $C'$ is an ancestor of $C$, and $f$ is a boundary vertex of $C$, or $r_C$. And if $C_1, \ldots, C_k$ are all the internal components of $T \setminus F$, then the number of those triples is bounded by $(|\partial(C_1)| + 1) \cdot d + \cdots + (|\partial(C_k)| + 1) \cdot d = (|\partial(C_1)| + \cdots + |\partial(C_k)| + k) \cdot d \leq (d + k) \cdot d \leq (d + d) \cdot d = O(d^2)$.

▶ **Proposition 11** ([6]). *Algorithm 1 correctly computes all Type-1 edges to construct a connectivity graph for the internal components of $T \setminus F$. The running time of this algorithm is $O(d^2 \log n)$.*

**Algorithm 1** Compute all Type-1 edges to construct a connectivity graph $\mathcal{R}$ for the internal components of $T \setminus F$.

```
1  foreach internal component C of T \ F do
2      let f_1, ..., f_k be the boundary vertices of C, sorted in increasing order
3      // process every internal component C' that is an ancestor of C
4      set f' ← p(r_C)
5      while f' ≠ ⊥ do
6          if p(f') ≠ parent_F(f') then
7              let C' be the internal component of T \ F with f' ∈ ∂(C')
8              set I ← [r_{C'}, p(f')]
9              if at least one of the following queries is positive:
10             2D_range([r_C, f_1 − 1] × I)
11             2D_range([f_1 + ND(f_1), f_2 − 1] × I)
12             ...
13             2D_range([f_{k−1} + ND(f_{k−1}), f_k − 1] × I)
14             2D_range([f_k + ND(f_k), r_C + ND(r_C) − 1] × I) then
15                 add the Type-1 edge (r_C, r_{C'}) to R
16             end
17         end
18         f' ← parent_F(f')
19     end
20 end
```

The construction of Type-2 edges is not so straightforward. For every failed vertex $f$, and every two internal components $C$ and $C'$, such that $C$ is an ancestor of $f$ and $C'$ is an ancestor of $C$, we would like to know whether there is a hanging subtree of $f$, from which stem a back-edge $e$ with an endpoint in $C$ and a back-edge $e'$ with an endpoint in $C'$. The straightforward way to determine this is the following. Let $b$ (resp., $b'$) be the boundary vertex of $C$ (resp., $C'$) that is an ancestor of $f$. Then, for every hanging subtree of $f$ with root $c$, we perform 2D-range-emptiness queries on the rectangles $[c, c + ND(c) - 1] \times [r_C, p(b)]$ and $[c, c + ND(c) - 1] \times [r_{C'}, p(b')]$. If both queries are positive, then we know that $C$ and $C'$ are connected in $G \setminus F$ through the hanging subtree with root $c$.

Obviously, this method is not efficient in general, because the number of hanging subtrees of $f$ can be very close to $n$. However, it is the basis for our more efficient method. The idea is to perform a lot of those queries at once, for large batches of hanging subtrees. More specifically, we perform the queries on *consecutive* hanging subtrees of $f$ (i.e., their roots are consecutive children of $f$), for which we know that the answer is positive on $C'$ (i.e., for every one of those subtrees, there certainly exists a back-edge that connects it with $C'$). In order for this idea to work, we have to rearrange properly the lists of children of all vertices. (Otherwise, the hanging subtrees of $f$ that are connected with $C'$ through a back-edge may not be consecutive in the list of children of $f$.) In effect, we maintain several DFS trees (specifically: $d_\star$), and several 2D-range-emptiness data structures, one for every different arrangement of the children lists.

Let us elaborate on this idea. Let $H$ be a hanging subtree of $f$ that connects some internal components, and let $C'$ be the lowest one among them (i.e., the one that is an ancestor of all the others). Then we have that the lower ends of all back-edges that stem from $H$ and end in ancestors of $C'$ are failed vertices that are ancestors of $C'$. Thus, since there are at

most $d$ failed vertices in total, we have that at least one among $low_1(r_H), \ldots, low_d(r_H)$ is in $C'$. In other words, $r_H$ is one of the children of $f$ whose $low_i$ point is in $C'$, for some $i \in \{1, \ldots, d\}$. Now, assume that for every $i \in \{1, \ldots, d_\star\}$, we have a copy of the list of the children of $f$ sorted in increasing order w.r.t. the $low_i$ point; let us call this list $L_i(f)$, and let it be stored in way that allows for binary search w.r.t. the $low_i$ point. Then, for every internal component $C$ that is an ancestor of $f$, we can find the segment $S_i(C)$ of $L_i(f)$ that consists of the children of $f$ whose $low_i$ point lies in $C$, by searching for the leftmost and the rightmost child in $L_i(f)$ whose $low_i$ point lies in $[r_C, p(b)]$, where $b$ is the boundary vertex of $C$ that is an ancestor of $f$.

Now let $i \in \{1, \ldots, d\}$ be such that $low_i(r_H) \in C'$. Then we have that $r_H \in S_i(C')$. Furthermore, we have that every child of $f$ that lies in $S_i(C')$ and is the root of a hanging subtree $H'$ of $f$ has the property that $H'$ is also connected with $C'$ through a back-edge. Thus, we would like to be able to perform 2D-range-emptiness queries as above on the subset $S$ of $S_i(C')$ that consists of roots of hanging subtrees, in order to determine the connectivity (in $G \setminus F$) of $C'$ with all internal components $C$ that are ancestors of $f$ and descendants of $C'$. We could do this efficiently if we had the guarantee that $S$ consists of large segments of consecutive children of $f$. We can accommodate for that during the preprocessing phase: for every $i \in \{1, \ldots, d_\star\}$, we perform a DFS of $T$, starting from $r$, where the adjacency list of every vertex $v$ is given by $L_i(v)$.[5] Let $T_i$ be the resulting DFS tree, and let $DFS_i$ be the corresponding DFS numbering. Then, with the DFS numbering of $T_i$, we initialize a data structure *2D_range_i*, for answering 2D-range-emptiness queries for back-edges w.r.t. $T_i$ in subrectangles of $[1, n] \times [1, n]$.

Now let us see how everything is put together. Let $H$ be a hanging subtree of $f$ that connects two internal components $C_1$ and $C_2$, and let $b_1$ and $b_2$ be the boundary vertices of $C_1$ and $C_2$, respectively, that are ancestors of $f$. Let $C'$ be the lowest internal component that is connected through a back-edge with $H$. Then there is an $i \in \{1, \ldots, d\}$ such that $low_i(r_H) \in C'$. Let $S$ be the maximal segment of $S_i(C')$ that contains $r_H$ and consists of roots of hanging subtrees, let $L$ be the minimum of $S$ and let $R$ be the maximum of $S$.[6] Then the 2D-range-emptiness queries on $[L, R + ND(R) - 1]_i \times [r_{C_1}, p(b_1)]_i$ and $[L, R + ND(R) - 1]_i \times [r_{C_2}, p(b_2)]_i$ with *2D_range_i* are both positive, and so we will add the edges $(r_{C_1}, r_{C'})$ and $(r_{C_2}, r_{C'})$ to $\mathcal{R}$. This will maintain in $\mathcal{R}$ the information that $C'$, $C_1$ and $C_2$, are connected with the same hanging subtree of $f$.

The algorithm that constructs enough Type-2 edges to make $\mathcal{R}$ a connectivity graph of the internal components of $T \setminus F$ is given in Algorithm 2. Our result is stated in Proposition 12.

▶ **Proposition 12** ([6]). *Algorithm 2 computes enough Type-2 edges to construct a connectivity graph $\mathcal{R}$ for the internal components of $T \setminus F$ (supposing that $\mathcal{R}$ contains all Type-1 edges). The running time of this algorithm is $O(d^4 \log n)$.*

## 3.5 Answering the queries

Assume that we have constructed a connectivity graph $\mathcal{R}$ for the internal components of $T \setminus F$, and that we have computed its connected components. Thus, given two internal components $C$ and $C'$, we can determine in constant time whether $C$ and $C'$ are connected in $G \setminus F$, by simply checking whether $r_C$ and $r_{C'}$ are in the same connected component of $\mathcal{R}$ (see Lemma 10).

[5] I.e., it is necessary that the vertices in the adjacency list of $v$ appear in the same order as in $L_i(v)$.
[6] Notice that, due to the construction of $T_i$, we have that $DFS_i(L)$ and $DFS_i(R)$ are also the minimum and the maximum, respectively, of $DFS_i(S)$.

**Algorithm 2** Compute enough Type-2 edges to construct a connectivity graph for the internal components of $T \setminus F$.

```
1  foreach failed vertex f do
2      // process all pairs of internal components that are ancestors of f
3      set f' ← parent_F(f)
4      while f' ≠ ⊥ do
5          let C' be the internal component with f' ∈ ∂(C')
6          // skip the following if C' does not exist, and go immediately
               to Line 26
7          foreach i ∈ {1, ..., d} do
8              let S_i be the collection of all maximal segments of L_i(f) that consist of
                roots of hanging subtrees with their low_i point in C'
9          end
10         // process all internal components C that are ancestors of f
               and descendants of C'
11         set f'' ← f
12         while f'' ≠ f' do
13             let C be the internal component with f'' ∈ ∂(C)
14             // skip the following if C does not exist, and go
                   immediately to Line 24
15             // check if C is connected with C' through at least one
                   hanging subtree of f
16             foreach i ∈ {1, ..., d} do
17                 foreach S ∈ S_i do
18                     let L ← min(S) and R ← max(S)
19                     if 2D_range_i([L, R + ND(R) − 1]_i × [r_C, p(f'')]_i) = true then
20                         add the Type-2 edge (r_C, r_C') to R
21                     end
22                 end
23             end
24             f'' ← parent_F(f'')
25         end
26         f' ← parent_F(f')
27     end
28 end
```

Now let $x, y$ be two vertices in $V(G) \setminus F$. In order to determine whether $x, y$ are connected in $G \setminus F$, we try to substitute $x, y$ with roots of internal components of $T \setminus F$, and then we reduce the query to those roots. Specifically, if $x$ (resp., $y$) belongs to an internal component $C$ of $T \setminus F$, then the connectivity between $x$ and $y$ is the same as that between $r_C$ and $y$ (resp., $x$ and $r_C$). Otherwise, if $x$ (resp., $y$) belongs to a hanging subtree $H$ of $T \setminus F$, then we try to find an internal component that is connected with $H$ through a back-edge. If such an internal component $C$ exists, then we can substitute $x$ (resp., $y$) with $r_C$. Otherwise, $x, y$ are connected in $G \setminus F$ if and only if they belong to the same hanging subtree of $T \setminus F$. This idea is shown in Algorithm 3.

▶ **Proposition 13** ([6]). *Given two vertices $x, y$ in $V(G) \setminus F$, Algorithm 3 correctly determines whether $x, y$ are connected in $G \setminus F$. The running time of Algorithm 3 is $O(d)$.*

**Algorithm 3** query$(x, y)$.

```
1  if x lies in an internal component C and y lies in an internal component C' then
2      if r_C is connected with r_C' in R then return true
3      return false
4  end
5  // at least one of x, y lies in a hanging subtree
6  if x lies in a hanging subtree H then
7      // check whether H is connected with an internal component through
           a back-edge
8      for i ∈ {1, ..., d} do
9          if low_i(r_H) ≠ ⊥ and low_i(r_H) ∉ F then
10             return query(low_i(r_H), y)
11         end
12     end
13     // there is no internal component that is connected with H in G \ F
14     if y lies in H then return true
15     return false
16 end
17 return query(y, x)
```

## References


1 Michael A. Bender and Martin Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004. doi:10.1016/j.tcs.2003.05.002.

2 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In *Proceedings of the 27th ACM Symposium on Computational Geometry,*, pages 1–10, 2011. doi:10.1145/1998196.1998198.

3 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.

4 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

5 Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. *SIAM J. Comput.*, 49(6):1363–1396, 2020. doi:10.1137/17M1146610.

6 Evangelos Kosinas. Connectivity queries under vertex failures: Not optimal, but practical. *arXiv version*, 2023. arXiv:2305.01756.

7 Yaowei Long and Thatchaphol Saranurak. Near-optimal deterministic vertex-failure connectivity oracles. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 1002–1010, 2022. doi:10.1109/FOCS54457.2022.00098.

8 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

9 Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In *49th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 229 of *LIPIcs*, pages 102:1–102:18, 2022. doi:10.4230/LIPIcs.ICALP.2022.102.

10 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

# Improved Approximations for Translational Packing of Convex Polygons

**Adam Kurpisz**
Bern University of Applied Sciences, Switzerland
Department of Mathematics, ETH Zürich, Switzerland

**Silvan Suter**
Department of Mathematics, ETH Zürich, Switzerland

## Abstract

Optimal packing of objects in containers is a critical problem in various real-life and industrial applications. This paper investigates the two-dimensional packing of convex polygons without rotations, where only translations are allowed. We study different settings depending on the type of containers used, including minimizing the number of containers or the size of the container based on an objective function.

Building on prior research in the field, we develop polynomial-time algorithms with improved approximation guarantees upon the best-known results by Alt, de Berg and Knauer, as well as Aamand, Abrahamsen, Beretta and Kleist, for problems such as Polygon Area Minimization, Polygon Perimeter Minimization, Polygon Strip Packing, and Polygon Bin Packing. Our approach utilizes a sequence of object transformations that allows sorting by height and orientation, thus enhancing the effectiveness of shelf packing algorithms for polygon packing problems. In addition, we present efficient approximation algorithms for special cases of the Polygon Bin Packing problem, progressing toward solving an open question concerning an $\mathcal{O}(1)$-approximation algorithm for arbitrary polygons.

**2012 ACM Subject Classification** Theory of computation → Packing and covering problems

**Keywords and phrases** Approximation algorithms, Packing problems, Convex polygons, Bin packing, Strip packing, Area minimization

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.76

**Related Version** *Full Version*: https://arxiv.org/abs/2308.08523

## 1 Introduction

Many real-life situations require us to make decisions about optimally packing a collection of objects into a specific container. One particular category of these packing problems is two-dimensional packing, which is encountered in everyday scenarios like arranging items on a shelf and in industrial applications such as cutting cookies from rolled-out dough or manufacturing sets of tiles from standard-sized panels made of wood, glass, or metal. Another intriguing example involves cutting fabric pieces for clothing production. In this case, the pieces often cannot be rotated freely, as they must adhere to a desired pattern in the final product, which is tailored of multiple elements. The widespread applicability of two-dimensional packing problems has led to a surge of interest in designing efficient algorithms to address them. In this paper, we follow the line of research and study the problem of packing convex polygons without rotations in various settings depending on the type of containers used.

Past research focusing on theoretical considerations of two-dimensional packings mainly concentrates on the scenario when all objects are axis-parallel rectangles. In this paper, we will discuss packing without rotations, in which only translations are permitted. There are two main classes of the problem depending on whether the size of the container is fixed and we want to minimize the number of containers used or whether we want to minimize the container's size with respect to some objective function.


A seminal example of the first class is the Geometric Bin Packing problem in which a number of unit size squared bins to pack is to be minimized. The problem is arguably the most natural generalization of the regular $(1D)$ Bin Packing to two dimensions, and its absolute approximability has been fully understood. Unless $\mathcal{P} = \mathcal{NP}$, the best possible efficient constant factor approximation is 2 [15], and such an algorithm is known [10].

In the second class, there are several variants to be considered. An example is the Strip Packing Problem which is concerned with packing objects into a strip of width 1 and infinite height in such a way that the maximum of all the heights of the placed objects is minimized. Like Geometric Bin Packing, Strip Packing generalizes $(1D)$ Bin Packing. The best known efficient constant factor approximation for Strip Packing has approximation factor $(5/3 + \epsilon)$ [9]. It is known that there can not exist a polynomial time algorithm with an approximation ratio of $(3/2 - \epsilon)$ for any $\epsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$, which follows directly from the approximation hardness of $(1D)$ Bin Packing. Both classes of problems have been also considered in the asymptotic setting, see e.g. [11, 5, 12].

In younger time, there was also an increase of interest in cases where the objects in question are general convex polygons. Alt, de Berg and Knauer [2, 3] considered the problem of packing an instance consisting of a number of convex polygons of the form $p \subset [0, 1]^2$ into a minimum area axis-parallel rectangular container. We refer to this problem as Polygon Area Minimization throughout this paper. In the special case where the instance consists of rectangles only, the problem is known to admit a PTAS [4]. They proved the existence of the following efficient algorithm:

- A 23.78-approximation for Polygon Area Minimization.

Recently, Aamand, Abrahamsen, Beretta and Kleist [1] showed that the algorithm of Alt, de Berg and Knauer can be leveraged to obtain also efficient approximation algorithms of further problems:

- A 7-approximation for Polygon Perimeter Minimization.
- A 51-approximation for Polygon Strip Packing.
- An 11-approximation for Polygon Bin Packing for polygons with diameter at most $\frac{1}{10}$.

## 1.1 Our results

The results of Alt, de Berg and Knauer [2, 3] and Aamand et al. [1] are heavily based on so-called shelf packing algorithms. In shelf packing algorithms, the objects are first placed on the shelves, possibly ordered by height, which are later stacked on one another to build a final solution. Compared to axis-parallel rectangles, the main challenge in designing an approximation algorithm for polygon packing problems is that objects cannot be sorted by height and orientation simultaneously. As a result, the algorithm and its analysis in [2, 3] have such a large approximation guarantee. In this paper, we provide new insight into how shelf packing algorithms should be applied to polygon packing problems. We introduce a sequence of transformations of the objects that allow us first to sort them by height and later by orientation to build a solution of a much better approximation guarantee. More precisely, we design polynomial-time algorithms with the following factors:

- A 9.45-approximation for Polygon Area Minimization.

Using this algorithm as a subroutine, we build upon the methods from Aamand et al. [1] to obtain the following efficient approximation algorithms:

- A $(3.75 + \epsilon)$-approximation for Polygon Perimeter Minimization.
- A 21.89-approximation for Polygon Strip Packing.
- A 5.09-approximation for Polygon Bin Packing for polygons which have their diameter bounded by $\frac{1}{10}$.

The results are proved in Sections 4, 5, 6, and 7 respectively. Furthermore, concerning Polygon Bin Packing, in the full version of this paper we show the following results, which make progress towards solving an open question of a $\mathcal{O}(1)$-approximation algorithm for Polygon Bin Packing for arbitrary polygons.

- There is an efficient $\mathcal{O}(\frac{1}{\delta})$-approximation algorithm for Polygon Bin Packing for instances where each polygon has width or height at most $1-\delta$.
- There is an efficient $\mathcal{O}(1)$-approximation algorithm for Polygon Bin Packing for instances with the property that all polygons share a spine (up to translation) with height at least $\frac{3}{4}$.

## 2 Preliminaries

We start out considerations by recalling a classical and well-known problem in theoretical computer science, the Bin Packing problem: Given a list of numbers $s_1, \ldots, s_n \in (0,1] \cap \mathbb{Q}$, representing the sizes of $n$ objects, the goal is to find the minimum number of bins of size 1, so that we can pack all objects into them. Bin Packing can be seen as the task of packing (1-dimensional) intervals into as few intervals of length 1 as possible. This definition can be extended to the two dimensional case. To do so we introduce several definitions.

Throughout the paper, for a set subset $A$ of the domain of a function $f$, $f(A)$ is a shorthand notation for $\sum_{a\in A} f(a)$. A **packing instance** is defined by a finite set $I$ containing objects to pack, a countable set $\mathcal{R}$ containing bins to pack the objects into and a set $\Phi$ of allowed transformations. A **shape** is a compact connected set $s \subset \mathbb{R}^2_{\geq 0}$. We denote a rectangular shape $r \subset \mathbb{R}^2_{\geq 0}$ by a tuple $r = (w,h) \in \mathbb{R}^2_{>0}$, which has width $w$ and height $h$. An **object** $o$ is an element of $I$ and has a shape $s \subset [0,1]^2$. Furthermore, we define $\mathcal{I}_{\square}$ and $\mathcal{I}_{\diamond}$ to be the sets which contain all finite sets $I$ with objects consisting of axis-parallel rectangles and convex polygons, respectively. We will usually denote $|I|$ by $n$. In order to ensure computability, we assume that each object in $\mathcal{I}_{\square}$ and $\mathcal{I}_{\diamond}$ is defined by finitely many vertices.

A **bin** $R \in \mathcal{R}$ is also characterized by having a shape. In this paper, we always assume that the shape of a bin $R$ is an axis-parallel rectangle. That is, it is a rectangle with each of its sides being parallel to one of the primal axes in $\mathbb{R}^2$, and normally, its lower left corner is at the origin $(0,0) \in \mathbb{R}^2$. We write $\texttt{width}(R)$ and $\texttt{height}(R)$ for the width and the height of a bin $R \in \mathcal{R}$. If $R = [0,1]^2$, we say that $R$ is a **unit bin**. In this study, the set of **allowed transformations** $\Phi$ is the set of all translations, i.e. $\Phi = \{\phi : R^2 \to \mathbb{R}^2 \mid \exists x_0 \in \mathbb{R}^2\ \forall x \in \mathbb{R}^2 : \phi(x) = x + x_0\}$. Note that we consider the setting where rotations or reflections of objects are not allowed. We denote the width and the height of an object $o$ (the length of the projection on the $x$-axis or $y$-axis respectively) by $\texttt{width}(o)$ and $\texttt{height}(o)$. The maximum width and height of a shape of an object in $I$, we denote by $w_{\max}(I)$ and $h_{\max}(I)$ respectively. We will also use the notation $\texttt{width}(S)$ and $\texttt{height}(S)$ for arbitrary subsets $S \subset \mathbb{R}^2$

A **packing** $P$ of $I$ is defined as a set of pairwise disjoint placements of all $o \in I$ with respect to the set of translations $\Phi$. Given a bin $R \in \mathcal{R}$, we say that we can **pack $I$ into $R$** if there is a packing $P$ of $I$ so that $P \subset R$. In this case, we also say that $P$ is a **packing of $I$ into $R$**. If we can pack the objects $I$ into the bin $R$, we call $R$ a **bounding box** of $I$. If we have an at most countable collection of bins $\mathcal{R} := \{R_j\}_{j\in J}$, we say that we can **pack $I$ into $\mathcal{R}$**, if there is a partition $I = \bigsqcup_{j\in J} I_j$ so that we can pack the objects $I_j$ into $R_j$ for all $j \in J$. In such case, if for all $j \in J$, $P_j$ is a packing of $I_j$ into $R_j$, we refer to $\mathcal{P} := \{P_j\}_j$ as a **multi-packing**. For any $j \in J$, we say that **$o$ is in the packing $P_j$**, if $o \in I_j$. We define the $\texttt{area}(P)$ to be the area of the smallest axis-parallel rectangle containing $P$.

The definition already demonstrates the more difficult nature of multi-dimensional packings compared to one dimensional packings. Even if the objects to pack are axis-parallel rectangles, it is no longer sufficient to consider in which bin to pack which rectangle, but the exact position in the bin also matters.

In this paper we consider problems consisting in packing convex polygonal shapes into axis-parallel rectangular bins under translational transformations. More precisely we consider the following problems.

▶ **Problem 1** (Polygon Packing).
***Input:*** *Convex polygons* $I \in \mathcal{I}_{\diamond}$.
***Goals:***

- *Bin Packing: Find the minimum number* $B \in \mathbb{N}$ *so that we can pack* $I$ *into* $B$ *unit bins.*
- *Strip Packing: Find the minimum height* $H \in \mathbb{Q}_{>0}$ *so that we can pack* $I$ *into a bin of width* 1 *and height* $H$*.*
- *Area Minimization: Find a bounding box* $R \in \mathbb{R}^2_{>0}$ *of* $I$ *so that* $f(R) = \texttt{width}(R) \cdot \texttt{height}(R)$ *is minimal.*
- *Perimeter Minimization: Find a bounding box* $R \in \mathbb{R}^2_{>0}$ *of* $I$ *so that* $f(R) = 2(\texttt{width}(R) + \texttt{height}(R))$ *is minimal.*
- *Minimum Square: Find a bounding box* $R \in \mathbb{R}^2_{>0}$ *so that* $f(R) = \max\{\texttt{width}(R), \texttt{height}(R)\}$ *is minimal.*

*Throughout the paper for the problems under consideration, we denote the optimal value of an instance* $I \in \mathcal{I}_{\diamond}$ *by* $\mathrm{opt}(I)$.

## 3 Shelf Packing Algorithms

Introducing well-known shelf-packing algorithms involves basic $(1D)$-Bin Packing algorithms, such as NextFit (NF), FirstFit (FF), and BestFit (BF). These place items $s_1, \ldots, s_n$ into bins sequentially, with $s_{i+1}$ placed according to specific rules. If no placement adheres to the rule, a new bin is opened. The rules differ for each algorithm.

- NF, places $s_{i+1}$ into the most recently opened bin, if it has enough space.
- FF, places $s_{i+1}$ in the earliest opened bin in which it fits.
- BF, places $s_{i+1}$ in the bin with least free space among bins in which $s_{i+1}$ fits.

NF and FF are often preprocessed by sorting the items in non-increasing size, which are called NextFitDecreasing (NFD) and FirstFitDecreasing (FFD). It is not hard to show that NF is a 2-approximation for the Bin Packing problem [14]. Moreover, NF packs it into at most $1 + 2\sum_{i=1}^{n} s_i$ bins. BF and FF both have an approximation ratio of 1.7 [7, 8], which are tight. Furthermore, as shown in [13], if $s_i \leq \frac{1}{m}$ for all $i \in [n]$ and some $m \geq 2$, FF packs this instance into at most $1 + (1 + \frac{1}{m})\sum_{i=1}^{n} s_i$ bins. [1]

Variants of NF, FF and BF exist for 2D rectangle packing problems, called NextFitDecreasingHeight (NFDH), FirstFitDecreasingHeight (FFDH), and BestFitDecreasingHeight (BFDH). These shelf-packing algorithms were introduced for Strip Packing, placing rectangles $r_1, \ldots, r_n \in \mathcal{I}_{\square}$ into a bin $R = [0, 1] \times [0, \infty)$ with infinite height. The three algorithms order rectangles $r_1, \ldots, r_n$ in non-increasing height and place them sequentially

[1] In fact they show that FF packs such instance in $2 + (1 + \frac{1}{m})\sum_{i=1}^{n} s_i$ bins. With a strategy analogous to the one from the proof of Theorem 3 in [6] in the two-dimensional case, however, one can show that an additive factor of 1 is sufficient.

into shelves in $R$. A shelf is a horizontal strip, and rectangles can open new shelves. A shelf-packing algorithm places $r_{i+1}$ in an existing shelf according to a rule or opens a new shelf. The placement is bottom-left without intersecting other rectangles.

- NFDH places $r_{i+1}$ in the most recently opened shelf, if it fits.
- FFDH places $r_{i+1}$ in the lowest possible shelf which has enough space.
- BFDH, as FFDH, allows for placing rectangles in a lower shelf than the most recently opened one. Here, $r_{i+1}$ is placed in the one which has minimal free horizontal space at the right end of the shelf among all shelves, while still having at least $\texttt{width}(r_{i+1})$ of it.

Since NFDH and FFDH are of particular interest in our paper, we present the following two absolute approximability results for axis-parallel rectangle Strip Packing problem.

▶ **Theorem 2** (Theorem 1 [6]). *Let $I \in \mathcal{I}_{\square}$. The packing $P$ obtained by* NFDH satisfies

$$\texttt{height}(P) \leq h_{\max}(I) + 2\,\texttt{area}(I) \leq 3\,\mathrm{opt}(I).$$

▶ **Theorem 3** (Theorem 3 [6]). *Let $m \in \mathbb{N}$ and $I \in \mathcal{I}_{\square}$ be satisfying that $w_{\max}(I) \leq \frac{1}{m}$. The packing $P$ of $I$ obtained by* FFDH satisfies

$$\texttt{height}(P) \leq h_{\max}(I) + \left(1 + \frac{1}{m}\right)\texttt{area}(I) \leq \left(2 + \frac{1}{m}\right)\mathrm{opt}(I).$$

## 4 An Efficient 9.45-Approximation for Polygon Area Minimization and 7-Approximation when all Polygons are $x$-Parallelograms

In this section, we prove that there is an efficient $9.\overline{4}$-approximation algorithm for Polygon Area Minimization, which improves the previous best approximation factor for polynomial time algorithms of 23.78 by Alt et al. [2, 3]. Based on that result, we also show a 7-approximation in the special case when all polygons are $x$-parallelograms, a special type of parallelograms we will introduce in Definition 5.

### 4.1 An Efficient 9.45-Approximation for Polygon Area Minimization

To start the discussions, we introduce the following definition.

▶ **Definition 4.** *Let $p \subset [0,1]^2$ be a polygon. A* ***spine*** *$s$ of $p$ is a (straight) line segment connecting a point in $\operatorname{argmin}_{(x,y)\in p} y$ with a point in $\operatorname{argmax}_{(x,y)\in p} y$. We call the angle between the $x$-axis in increasing direction and $s$ the* ***angle of $s$****. We say that $s$* ***is tilted to the right*** *or* ***leans to the right****, if this angle is in $(0, \frac{\pi}{2}]$ and* ***is tilted to the left*** *or* ***leans to the left****, if it is in $[\frac{\pi}{2}, \pi)$.*

Sometimes we also talk about "the" spine of a polygon, implicitly assuming that one has been fixed. The algorithm of Alt et al. is a shelf-packing algorithm and ordering polygons by the angle of their spines is a crucial step. Our algorithm shares these two characteristics. A key difference is that our algorithm first packs the polygons into parallelograms that have two of their sides parallel to the $x$-axis. We give such parallelograms their own definition:

▶ **Definition 5.** *An* ***$x$-parallelogram*** *is a parallelogram $q \subset \mathbb{R}^2$ that has two of its sides parallel to the $x$-axis. Of those two sides, we call the one with lower $y$-coordinate the* ***base*** *of $q$. We write $\texttt{base}(q)$ for the length of the base of $q$ and $\texttt{wside}(q)$ for the width of one of the sides of $q$ which is not parallel to the $x$-axis. Similar as to the definition for spines of*

*polygons, we say that $q$* ***is tilted to the right*** *or* ***leans to the right****, if the angle between its right side and the increasing direction of the $x$-axis is in $(0, \frac{\pi}{2}]$ and* ***is tilted to the left*** *or* ***leans to the left****, if it is in $[\frac{\pi}{2}, \pi)$. We refer to this angle simply as* ***angle of $q$****.*

For packing polygons into $x$-parallelograms, we prove the following result, which is a refined version of the discussions of Aamand et al. [1] in Subsection 5.2.4 of their paper.

▶ **Lemma 6.** *Let $p$ be a convex polygon. Then there exists an $x$-parallelogram $q$ that contains $p$ and satisfies*

**(i)** $\texttt{height}(q) = \texttt{height}(p)$
**(ii)** $\texttt{base}(q) \leq \texttt{width}(p)$
**(iii)** $\texttt{wside}(q) \leq \texttt{width}(p)$
**(iv)** $\texttt{area}(q) \leq 2\,\texttt{area}(p)$.

**Proof.** First, we construct a bounding $x$-parallelogram as the one that can be seen in Figure 1. Let $l_b$ and $l_t$ be lines parallel to the $x$-axis and tangent to $p$, touching the bottom of $p$ and the top of $p$ respectively. Choose $p_b \in p \cap l_b$ and $p_t \in p \cap l_t$ and define $s$ to be the line connecting $p_b$ and $p_t$. Note that $s$ is a spine of $p$. Let $s_l$ and $s_r$ be tangent to $p$ and parallel to $s$, lying on the left and on the right of $p$. Let $p_l \in p \cap s_l$ and $p_r \in p \cap s_r$. We now define $q$ to be the set bounded by $l_b, l_t, s_l$ and $s_r$. Note that $q$ is an $x$-parallelogram that satisfies $(i)$. As the left and right sides of $q$ are just translations of $s$, it also satisfies $(iii)$, because of course $s \subset p$ by convexity of $p$.

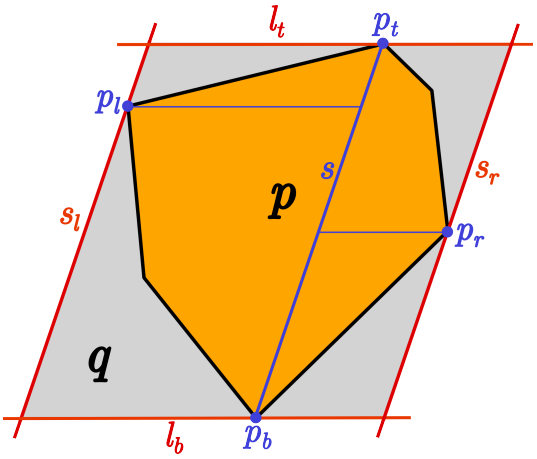



**Figure 1** The construction of a bounding parallelogram $q$ from the proof of Lemma 6 on an example polygon $p$. Note that here, it holds that $\texttt{base}(q) > \texttt{width}(p)$.

To see that $q$ also satisfies $(iv)$, we consider the triangle with vertices $p_t, p_b$ and $p_l$. We note that it is contained in $p$ due to convexity and contains exactly half the area of the part of $q$ that lies on the left of $s$. Analogously, the triangle with vertices $p_t, p_b$ and $p_r$ has half the area of the part of $q$ that lies on the right of $s$. So $q$ does indeed also satisfy $(iv)$.

However, $q$ does not necessarily satisfy $(ii)$ (see the polygon in Figure 1). If it does, then $q$ satisfies all assumptions of the lemma and we are done.

So we consider now the case when $\texttt{base}(q) > \texttt{width}(p)$. Consider the axis-parallel rectangle $r$ bounding $p$. That is, $r$ contains $p$ and each of its sides has non-empty intersection with $p$. Surely $r$ satisfies $(i), (ii)$ and $(iii)$. To see that it also satisfies $(iv)$, note that

$$\begin{aligned}\texttt{area}(r) &= \texttt{width}(r)\,\texttt{height}(r)\\ &= \texttt{width}(p)\,\texttt{height}(p)\\ &< \texttt{base}(q)\,\texttt{height}(q)\\ &= \texttt{area}(q)\\ &\leq 2\,\texttt{area}(p).\end{aligned}$$

So whenever $\texttt{base}(q) > \texttt{base}(p)$, we showed that $r$ satisfies all requirements $(i) - (iv)$ of the lemma instead. Since $r$ is also an $x$-parallelogram, we conclude. ◀

With the help of Lemma 6, we can now present the main result of this chapter.

▶ **Theorem 7.** *There is a polynomial time $9.\overline{4}$-approximation algorithm for Polygon Area Minimization. Moreover, there is such algorithm with running time $\mathcal{O}(n^2 + N)$, where $n$ is the number of polygons and $N$ the total number of vertices in a given input $I \in \mathcal{I}_{\diamondsuit}$, assuming that each polygon $p \in I$ is given as a list of its vertices.*

**Proof.** Let $I \in \mathcal{I}_{\diamondsuit}$. We construct a packing $P$ as the one depicted in Figure 2.

Pack each polygon $p \in I$ into an $x$-parallelogram $q$ as in Lemma 6. Call the instance of all so-obtained $x$-parallelograms $I_Q$. The idea of our algorithm is to use FFDH to pack straightened, axis-parallel versions of the $x$-parallelograms $I_Q$ and to then use this packing to obtain one for $I_Q$ and hence also $I$ which is not much bigger.

So, define yet another instance $I_R$ which, for each $q \in I_Q$, contains a rectangle $r = (\texttt{base}(q), \texttt{height}(q))$. With FFDH, we now pack $I_R$ into a strip of width $cw_{\max}(I)$, where $c \geq 1$ is to be determined later. Call the so-obtained packing $P_R$. By Theorem 3 it follows

$$\texttt{height}(P_R)(cw_{\max}(I)) \leq \left(1 + \frac{1}{m}\right)\texttt{area}(I_R) + ch_{\max}(I_R)w_{\max}(I), \tag{1}$$

where $m = \lfloor c \rfloor$, as $w_{\max}(I_R) = \max_{q \in I_Q} \texttt{base}(q) \leq w_{\max}(I)$ by Lemma 6.

Let $S \subset I_Q$ be the parallelograms corresponding to the rectangles in a certain shelf in the packing $P_R$. We can pack $S$ into a new shelf of width $(c+2)w_{\max}(I)$ by first ordering the parallelograms $S$ by decreasing angle. Indeed, if we, after this ordering, place them all next to each other in the shelf, we note that now all bases of the parallelograms are connected to each other and hence the overlap on either side is at most $\max_{q \in S} \texttt{wside}(q) \leq w_{\max}(I)$ by Lemma 6. Put all such shelves on top of each other and call the so-obtained packing $P_Q$. Note that $P_Q$ has the same height as $P_R$, but $\frac{c+2}{c}$ times its width. Finally, we pack each polygon into its respective parallelogram in the packing $P_Q$. Call this packing $P$. Then

$$\begin{aligned}\texttt{area}(P) &\leq \texttt{area}(P_Q)\\ &\leq \frac{c+2}{c}\,\texttt{height}(P_R)(cw_{\max}(I))\\ &\leq \frac{c+2}{c}\left(1 + \frac{1}{m}\right)\texttt{area}(I_R) + (c+2)h_{\max}(I_R)w_{\max}(I)\\ &= \frac{c+2}{c}\frac{m+1}{m}\,\texttt{area}(I_Q) + (c+2)h_{\max}(I_Q)w_{\max}(I)\\ &\leq 2\frac{c+2}{c}\frac{m+1}{m}\,\texttt{area}(I) + (c+2)h_{\max}(I)w_{\max}(I)\\ &\leq \left(2\frac{c+2}{c}\frac{m+1}{m} + (c+2)\right)\mathrm{opt}(I).\end{aligned}$$

One can check that the minimum is attained at $c = 3$, in which case we get a $9.\overline{4}$-approximation.

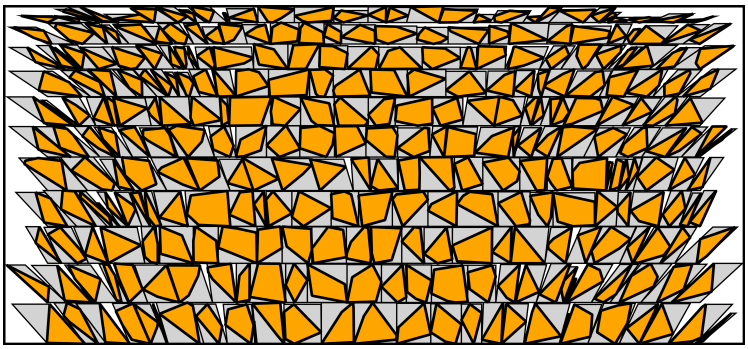


**Figure 2** A packing computed with the algorithm from the proof of Theorem 7, here with parameter $c = 15$. For each polygon, also its computed bounding $x$-parallelogram is drawn.

We now note that the claimed running times of the above algorithm follows by observing that: Constructing $I_Q$ from $I$ can be done in $\mathcal{O}(N)$ time, constructing $I_R$ from $I_Q$ can be done in $\mathcal{O}(n)$ time, constructing $P_R$ can be done in $\mathcal{O}(n^2)$ time, constructing $P_Q$ from $P_R$ can be done by sorting in $\mathcal{O}(n \log(n))$ time, and finally, constructing $P$ from $P_Q$ can be done in $\mathcal{O}(N)$ time. ◀

The algorithm of Alt et al. runs in time $\mathcal{O}(N \log(N))$ and thus for certain instances faster than our algorithm. If, in our algorithm, the use of FFDH for packing $I_R$ is replaced by NFDH, one can show using Theorem 2 that for an optimal choice of $c = 2\sqrt{2}$, the algorithm has an approximation guarantee of 11.66 while having a running time of $\mathcal{O}(n \log(n) + N)$, thus obtaining an algorithm that runs faster than the algorithm of Alt et al., while still having a greatly improved approximation ratio.

## 4.2 An Efficient 7-Approximation for Polygon Area Minimization when all Polygons are $x$-Parallelograms

In the algorithm presented in the previous subsection, we first pack general polygons into $x$-parallelograms and afterwards pack these $x$-parallelograms. One would expect that if one wants to pack $x$-parallelograms from the start, one should be able to obtain a better approximation factor. Showing that this is indeed the case is the content of this brief section.

▶ **Theorem 8.** *There is a polynomial time* 7*-approximation algorithm for* Polygon Area Minimization*, when all input polygons are $x$-parallelograms.*

The proof follows along the lines of the proof of Theorem 7.

**Proof.** Let $I \in \mathcal{I}_{\diamond}$ be so that every $p \in I$ is an $x$-parallelogram. As in the proof of Theorem 7, we define the set $I_R \in \mathcal{I}_{\square}$ that for every $p \in I$ contains some $r \in I_R$ with $\texttt{width}(r) = \texttt{base}(p)$ and $\texttt{height}(r) = \texttt{height}(p)$. Again, we pack $I_R$ with FFDH into a strip of width $cw_{\max}(I)$ and call the so-obtained packing $P_R$.

After ordering them by angle, we can pack the parallelograms $S \subset I$ corresponding to the rectangles in some shelf in $P_R$ into a new shelf of width $(c+2)w_{\max}(I)$, because of course $\texttt{wside}(p) \leq w_{\max}(I)$. Therefore, calling the so-obtained packing $P$,

$$\begin{aligned}\texttt{area}(P) &\leq \frac{c+2}{c}\,\texttt{area}(P_R)\\ &\leq \frac{c+2}{c}\frac{m+1}{m}\,\texttt{area}(I) + (c+2)h_{\max}(I)w_{\max}(I)\\ &\leq \left(\frac{c+2}{c}\frac{m+1}{m} + (c+2)\right)\operatorname{opt}(I),\end{aligned}$$

which, for $c = 2$, is minimized and equal to 7. ◀

## 5 An Efficient $(3.75+\epsilon)$-Approximation for Polygon Perimeter Minimization and $(3.56+\epsilon)$-Approximation for Polygon Minimum Square

In this section, we show how to leverage the algorithm from Theorem 7 to obtain an approximation-algorithm for Polygon Perimeter Minimization. We improve on the polynomial time 7.3-approximation algorithm obtained by Aamand et al. [1] and present an efficient $(3.75+\epsilon)$-approximation. Moreover, we show how to leverage that result to obtain an $(3.56+\epsilon)$-approximation-algorithm for Polygon Minimum Square.

### 5.1 An Efficient $(3.75+\epsilon)$-Approximation for Polygon Perimeter Minimization

▶ **Theorem 9.** *For every $\epsilon > 0$, there is an efficient $(3.75+\epsilon)$-approximation algorithm for Polygon Perimeter Minimization.*

**Proof.** Let $I \in \mathcal{I}_{\diamond}$. Note that for the perimeter objective, it surely holds that

$$\operatorname{opt}(I) \geq 2(w_{\max}(I) + h_{\max}(I)). \tag{2}$$

Furthermore, since a bounding box of $I$ has area at least $\texttt{area}(I)$ and the minimum perimeter rectangle having area $\texttt{area}(I)$ is a square, it also is true that

$$\operatorname{opt}(I) \geq 4\sqrt{\texttt{area}(I)}.$$

It follows from Inequality 2 that

$$\min\{h_{\max}(I), w_{\max}(I)\} \leq \frac{1}{4}\operatorname{opt}(I)$$

and without loss of generality, we assume that $w_{\max}(I) \leq \frac{1}{4}\operatorname{opt}(I)$. Otherwise we are making the following argument by packing into vertical shelves instead.

Let $P$ be the packing obtained from the algorithm in Theorem 7, leaving $c$ as a free parameter. Then $P$ satisfies

$$\texttt{width}(P) \leq (c+2)w_{\max}(I)$$

and by dividing the inequality (1) by the width of the strip used for the rectangle packing obtained by FFDH, $cw_{\max}(I)$, we see that

$$\texttt{height}(P) \leq 2\frac{m+1}{m}\frac{\texttt{area}(I)}{cw_{\max}(I)} + h_{\max}(I) \leq \frac{1}{8}\frac{m+1}{m}\frac{\operatorname{opt}(I)^2}{cw_{\max}(I)} + h_{\max}(I).$$

We restrict the domain of $c$ so that $c \geq \frac{\operatorname{opt}(I)}{4w_{\max}(I)} \geq 1$ and write $c = l\frac{\operatorname{opt}(I)}{4w_{\max}(I)}$ for some $l \geq 1$. Then

$$\texttt{width}(P) \leq \frac{l}{4}\operatorname{opt}(I) + 2w_{\max}(I), \quad \texttt{height}(P) \leq \frac{1}{2l}\frac{m+1}{m}\operatorname{opt}(I) + h_{\max}(I).$$

The perimeter of $P$ is hence bounded by

$$\begin{aligned}
2(\texttt{width}(P) + \texttt{height}(P)) &\leq 2\left(\left(\frac{l}{4} + \frac{m+1}{2lm}\right)\operatorname{opt}(I) + 2w_{\max}(I) + h_{\max}(I)\right) \\
&\leq 2\left(\frac{1}{4}\left(l + \frac{2(m+1)}{lm}\right) + 1\right)\operatorname{opt}(I) \\
&= \frac{1}{2}\left(l + \frac{2(m+1)}{lm} + 4\right)\operatorname{opt}(I),
\end{aligned}$$

which is minimized and equal to $3.75\operatorname{opt}(I)$ when $l$ is equal to $\bar{l} := 2$, where we used that $m = \lfloor c \rfloor \geq \lfloor l \rfloor$.

Now since the value of $\operatorname{opt}(I)$ is not known beforehand and since $\bar{l}$ depends on it, we need to guess an optimal value for $l$. This can be done as follows. Compute the packing from the algorithm in Theorem 7 for $c = 1, (1+\epsilon), \ldots, (1+\epsilon)^K$, where $K = \frac{\log(n)}{\log(1+\epsilon)}$ and denote the packing obtained for $c = (1+\epsilon)^k$ by $P_k$ for all $k \in \{0, 1, \ldots, n\}$. Over all those packings, choose the one that has minimum perimeter. Say this perimeter is $z > 0$. Let $k \in \{1, \ldots, K\}$ be so that

$$(1+\epsilon)^{k-1} \leq \bar{c} \leq (1+\epsilon)^k,$$

where $\bar{c} := \bar{l}\frac{\operatorname{opt}(I)}{4w_{\max}(I)}$. Then, for $l_k := (1+\epsilon)^k \frac{4w_{\max}(I)}{\operatorname{opt}(I)}$, it holds that

$$l_k = (1+\epsilon)^k \frac{4w_{\max}(I)}{\operatorname{opt}(I)} \leq (1+\epsilon)\bar{c}\frac{4w_{\max}(I)}{\operatorname{opt}(I)} = (1+\epsilon)\bar{l}.$$

In particular, as of course also $\bar{l} \leq l_k$, it holds that

$$\begin{aligned}
z &\leq 2(\texttt{width}(P_k) + \texttt{height}(P_k)) \\
&\leq \frac{1}{2}\left(l_k + \frac{2(m+1)}{l_k m} + 4\right)\operatorname{opt}(I) \\
&\leq (1+\epsilon)\frac{1}{2}\left(\bar{l} + \frac{2(m+1)}{\bar{l}m} + 4\right)\operatorname{opt}(I) \\
&\leq (1+\epsilon)3.75\operatorname{opt}(I),
\end{aligned}$$

which shows the statement. ◀

## 5.2 An Efficient $(3.56+\epsilon)$-Approximation for Polygon Minimum Square

In this section, we show how to leverage the algorithm from Theorem 7 to obtain an approximation-algorithm for Polygon Minimum Square.

▶ **Theorem 10.** *For every $\epsilon > 0$, there is an efficient $(3.56+\epsilon)$-approximation algorithm for Polygon Minimum Square.*

The proof is similar to the proof of Theorem 9.

**Proof.** Let $I \in \mathcal{I}_{\diamondsuit}$. Note that

$$\operatorname{opt}(I) \geq \max\{w_{\max}(I), h_{\max}(I)\}$$

and also

$$\operatorname{opt}(I) \geq \sqrt{\texttt{area}(I)}.$$

Analogously to the proof of Theorem 9, but substituting $c = l\frac{\operatorname{opt}(I)}{w_{\max}(I)}$ for some $l \geq 1$ instead, we can construct a packing $P$ with

$$\texttt{width}(P) \leq l \operatorname{opt}(I) + 2w_{\max}(I), \quad \texttt{height}(P) \leq \frac{2}{l}\frac{m+1}{m}\operatorname{opt}(I) + h_{\max}(I).$$

Then

$$\max\{\texttt{width}(P), \texttt{height}(P)\} \leq \max\left\{l+2, \frac{2(m+1)}{lm} + 1\right\}\operatorname{opt}(I).$$

The minimum is attained for $l = \bar{l} := \frac{1}{2}(\sqrt{17}-1)$ in which case $m = 1$ and the approximation factor is equal to to $\frac{1}{2}(\sqrt{17}+3) \approx 3.56$. As in the proof of Theorem 9, we can guess the value of $\bar{l}$ to obtain a $(3.56+\epsilon)$-approximation algorithm. ◀

## 6 An Efficient 21.89-Approximation for Polygon Strip Packing

In this section, we show how to obtain a polynomial time $21.\overline{8}$-approximation for Polygon Strip Packing, improving on the previous best known approximation factor for efficient algorithms of 51 by Aamand et al. [1]. The idea is to construct vertical shelves with the algorithm from Theorem 7 and then to stack such vertical shelves horizontally into the strip. This idea comes from [1]. We slightly improve their procedure using the following observation.

▶ **Lemma 11.** *Let $\overline{I} \in \mathcal{I}_{\diamondsuit}$ and let $\overline{P} \subset [0,(c+2)w_{\max}(\overline{I})] \times [0,\infty)$ be a shelf-packing obtained by the algorithm from Theorem 7 for some $c \geq 1$. Let $I \subset \overline{I}$ be the polygons in one of the shelves of $\overline{P}$ and define the packing $P := \{\overline{P}(p)\}_{p\in I}$. Then there is a shelf-packing $P'$ of $I$ into two shelves with $\texttt{width}(P') \leq \frac{1}{2}(c+3)w_{\max}(\overline{I})$ and $\texttt{height}(P') \leq 2\,\texttt{height}(P)$.*

**Proof.** The idea is to simply split the shelf in half in its middle as follows. Let $x_{\text{mid}} := \frac{(c+2)w_{\max}(\overline{I})}{2}$ and partition $I$ into the two sets

$$I_L := \left\{p \in I \mid \texttt{width}\left(P(p) \cap (\mathbb{R}_{\geq x_{\text{mid}}} \times \mathbb{R})\right) \leq \frac{\texttt{width}(p)}{2}\right\}$$

and $I_R = I \backslash I_L$. Note that

$$I_R \subset \left\{p \in I \mid \texttt{width}\left(P(p) \cap (\mathbb{R}_{\leq x_{\text{mid}}} \times \mathbb{R})\right) < \frac{\texttt{width}(p)}{2}\right\}.$$

We can now pack $I_L$ and $I_R$ into separate shelves while respecting the ordering of the polygons in $P$. We denote the packing where both of those shelves are stacked on each other by $P'$. Both shelves have width at most

$$x_{\text{mid}} + \frac{w_{\max}(I)}{2} \leq x_{\text{mid}} + \frac{w_{\max}(\overline{I})}{2} = \frac{1}{2}(c+3)w_{\max}(\overline{I}).$$

and hence $P'$ does as well. ◀

Making use of the lemma, we can show the following result.

▶ **Theorem 12.** *There is a polynomial time* $21.\overline{8}$*-approximation algorithm for* Polygon Strip Packing.

**Proof.** Let $I \in \mathcal{I}_{\diamond}$. We apply the algorithm from Theorem 7, for some $c$ which we will fix later, to construct vertical shelves. More precisely, we rotate each item by an angle of $\pi/2$, apply the algorithm to the rotated instance, and then rotate the whole packing back by $\pi/2$. Let $P$ be the so obtained packing and let $S_1, \dots, S_k$ be the vertical shelves of this packing.

We now stack $S_1, \dots, S_k$ horizontally into the (vertical) strip. That is, we pack the $\{S_i\}_{i\in[k]}$ into horizontal shelves $T_1, \dots, T_l$ themselves. Note that since each shelf $S_i$, where $i \in [k]$, has the same height, the problem reduces to $(1D)$-Bin Packing. In particular, stacking the $\{S_i\}_{i\in[k]}$ greedily, each except for possibly the last horizontal shelf is covered by at least half by the $S_i$. If the last shelf $T_l$ is covered by half, we need at most $l \leq \frac{2\,\mathtt{area}(P)}{(c+2)h_{\max}(I)}$ horizontal shelves to pack $S_1, \dots, S_k$. Otherwise, we need at most $l \leq \frac{2\,\mathtt{area}(P)}{(c+2)h_{\max}(I)} + 1$ shelves. However, in this case we can reduce the height of $T_l$, which is filled less than half by the $\{S_i\}_{i\in[k]}$, to $\frac{1}{2}(c+3)h_{\max}(I)$ by using Lemma 11 on every shelf $S$ in $T_l$.

In particular, for such packing $P'$ it holds that

$$\begin{aligned}
\mathtt{height}(P') &\leq \left(\frac{2\,\mathtt{area}(P)}{(c+2)h_{\max}(I)}\right)(c+2)h_{\max}(I) + \frac{1}{2}(c+3)h_{\max}(I) \\
&= 2\,\mathtt{area}(P) + \frac{1}{2}(c+3)h_{\max}(I) \\
&\leq \left(4\frac{c+2}{c}\frac{m+1}{m} + \frac{5}{2}c + \frac{11}{2}\right)\mathrm{opt}(I) \\
&= 21.\overline{8}\,\mathrm{opt}(I),
\end{aligned}$$

which is attained for $c = 3$. ◀

## 7 Efficient Approximation Algorithms for Polygon Bin Packing for Polygons of Width Upper Bounded by $1/M$

In this section, we present polynomial time approximation algorithms for Polygon Bin Packing for the case when polygons have their width bounded by a fraction of the form $\frac{1}{M}$ and also improved approximations if their height is bounded as well.

To the best of our knowledge, the only presently published polynomial time approximation algorithm for Polygon Bin Packing is an 11-approximation in the case when the diameter of the input polygons is bounded by $\frac{1}{10}$, see [1]. For this case, we obtain an efficient 5.09-approximation.

We prove the following theorem.

▶ **Theorem 13.** *Let* $I \in \mathcal{I}_{\diamond}$ *and assume that there is some* $M \in \mathbb{N}_{\geq 2}$ *with* $w_{\max}(I) \leq \frac{1}{M}$. *Then one can pack* $I$ *into*

$$\begin{cases} 32\,\mathrm{opt}(I) + 5 & \text{if } M = 2 \\ \frac{4M(M-1)}{(M-2)^2}\,\mathrm{opt}(I) + 3 & \text{if } M \geq 3 \end{cases}$$

*bins efficiently. If additionally* $h_{\max}(I) \leq \frac{1}{M}$, *then we can even efficiently pack* $I$ *into*

$$\begin{cases} 24\,\mathrm{opt}(I) + 3 & \text{if } M = 2 \\ \frac{2(M+1)(M-1)}{(M-2)^2}\,\mathrm{opt}(I) + 2 & \text{if } M \geq 3 \end{cases}$$

*bins.*

In particular, for inputs where both width and height of all polygons are upper bounded by $\frac{1}{10}$, we get a 5.09-approximation.

**Proof.** Assume first that $M \geq 3$. We use the algorithm from Theorem 7 for $c = \frac{1}{w_{\max}(I)} - 2 \geq M - 2$ to obtain a packing $P$ with shelves $S_1, \ldots, S_k$. The packing $P$ satisfies

$$\texttt{width}(P) \leq (c+2)w_{\max}(I) = 1,$$

and hence a shelf $S_i$, $i \in [k]$, fits into a unit bin. Note that $m = \lfloor c \rfloor \geq M - 2$, since $M - 2$ is an integer. Since $cw_{\max}(I) = 1 - 2w_{\max}(I) \geq \frac{M-2}{M}$, it holds that

$$\texttt{height}(P) \leq 2\left(1 + \frac{1}{m}\right)\frac{\texttt{area}(I)}{cw_{\max}(I)} + h_{\max}(I) \leq 2\left(1 + \frac{1}{M-2}\right)\frac{\texttt{area}(I)}{(M-2)/M} + h_{\max}(I).$$

We can now use NF to distribute the shelves into unit bins. Since $h_{max} = 1$, this uses at most $2\,\texttt{height}(P) + 1$ bins, see Section 3. But

$$2\,\texttt{height}(P) + 1 \leq 4\left(1 + \frac{1}{M-2}\right)\frac{\texttt{area}(I)}{(M-2)/M} + 2h_{\max}(I) + 1 \leq \frac{4M(M-1)}{(M-2)^2}\,\mathrm{opt}(I) + 3.$$

If $h_{\max}(I) \leq \frac{1}{M}$ as well, we can use FF instead of NF to distribute the shelves into bins, which needs at most $(1 + \frac{1}{M})\,\texttt{height}(P) + 1$ bins, see again Section 3. This way, the number of needed bins is bounded by

$$\left(1 + \frac{1}{M}\right)\texttt{height}(P) + 1 \leq \frac{2(M+1)(M-1)}{(M-2)^2}\,\mathrm{opt}(I) + 2,$$

as claimed.

Now we consider the case when $M = 2$. Partition $I$ into two sets $I_L$ and $I_R$, where a polygon $p \in I$ belongs to $I_L$ if its chosen $x$-parallelogram in the proof of Theorem 7 is tilted to the left and to $I_R$ if it is tilted to the right. We now proceed with the algorithm in the proof of Theorem 7 for $c = \frac{1}{w_{max}(I)} - 1 \geq 1$, but for $I_L$ and $I_R$ separately. Call the obtained packings $P_L$ and $P_R$, respectively. Note that since all are tilted to one side only,

$$\texttt{width}(P_L), \texttt{width}(P_R) \leq (c+1)w_{\max}(I) = 1$$

and, using $m = |c| \geq 1$ and $cw_{\max}(I) = 1 - w_{\max}(I) \geq \frac{1}{2}$,

$$\texttt{height}(P_L) \leq 2\left(1 + \frac{1}{m}\right)\frac{\texttt{area}(I_L)}{cw_{\max}(I_L)} + h_{\max}(I_L) \leq 8\,\texttt{area}(I_L) + h_{\max}(I),$$

as well as analogously $\texttt{height}(P_R) \leq 8\,\texttt{area}(I_R) + h_{\max}(I)$.

Hence, for the packing $P$, where $P_R$ is stacked on top of $P_L$, it holds that

$$\texttt{width}(P) \leq 1 \quad \text{and} \quad \texttt{height}(P) \leq 16\,\texttt{area}(I) + 2h_{\max}(I).$$

From here, with the same arguments as for $M \geq 3$, we obtain the desired bounds. ◀

## References


1 Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1806–1833, 2023. doi:10.1137/1.9781611977554.ch69.

2 Helmut Alt, Mark de Berg, and Christian Knauer. Approximating minimum-area rectangular and convex containers for packing convex polygons. *Journal of Computational Geometry*, 8(1):1–10, 2017. doi:10.20382/jocg.v8i1a1.

3 Helmut Alt, Mark de Berg, and Christian Knauer. Corrigendum to: Approximating minimum-area rectangular and convex containers for packing convex polygons. *Journal of Computational Geometry*, 11(1):653–655, 2020. doi:10.20382/jocg.v11i1a26.

4 Nikhil Bansal, José R. Correa, Claire Kenyon, and Maxim Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006. doi:10.1287/moor.1050.0168.

5 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 13–25, 2014. doi:10.1137/1.9781611973402.2.

6 E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980. doi:10.1137/0209062.

7 György Dósa and Jiří Sgall. First Fit bin packing: A tight analysis. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, pages 538–549, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2013.538.

8 György Dósa and Jiří Sgall. Optimal analysis of best fit bin packing. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 429–441, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-43948-7_36.

9 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A $(5/3+\epsilon)$-approximation for strip packing. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, pages 475–487, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-22300-6_40.

10 Rolf Harren and Rob van Stee. An absolute 2-approximation algorithm for two-dimensional bin packing, 2009. doi:10.48550/arXiv.0903.2265.

11 Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2625, 2017. doi:10.1137/1.9781611974782.172.

12 Klaus Jansen and Roberto Solis-Oba. Rectangle packing with one-dimensional resource augmentation. *Discrete Optimization*, 6(3):310–323, 2009. doi:10.1016/j.disopt.2009.04.001.

13 D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974. doi:10.1137/0203025.

14 D.S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

15 Joseph Y-T. Leung, Tommy W. Tam, C.S. Wong, Gilbert H. Young, and Francis Y.L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990. doi:10.1016/0743-7315(90)90019-L.

# Structural Parameterizations for Two Bounded Degree Problems Revisited

**Michael Lampis**
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

**Manolis Vasilakis**
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016, Paris, France

## Abstract

We revisit two well-studied problems, Bounded Degree Vertex Deletion and Defective Coloring, where the input is a graph $G$ and a target degree $\Delta$ and we are asked either to edit or partition the graph so that the maximum degree becomes bounded by $\Delta$. Both problems are known to be parameterized intractable for the most well-known structural parameters, such as treewidth.

We revisit the parameterization by treewidth, as well as several related parameters and present a more fine-grained picture of the complexity of both problems. In particular:

- Both problems admit straightforward DP algorithms with table sizes $(\Delta+2)^{\mathrm{tw}}$ and $(\chi_{\mathrm{d}}(\Delta+1))^{\mathrm{tw}}$ respectively, where tw is the input graph's treewidth and $\chi_{\mathrm{d}}$ the number of available colors. We show that, under the SETH, both algorithms are essentially optimal, for any non-trivial fixed values of $\Delta, \chi_{\mathrm{d}}$, even if we replace treewidth by pathwidth. Along the way, we obtain an algorithm for Defective Coloring with complexity quasi-linear in the table size, thus settling the complexity of both problems for treewidth and pathwidth.
- Given that the standard DP algorithm is optimal for treewidth and pathwidth, we then go on to consider the more restricted parameter tree-depth. Here, previously known lower bounds imply that, under the ETH, Bounded Vertex Degree Deletion and Defective Coloring cannot be solved in time $n^{o(\sqrt[4]{\mathrm{td}})}$ and $n^{o(\sqrt{\mathrm{td}})}$ respectively, leaving some hope that a qualitatively faster algorithm than the one for treewidth may be possible. We close this gap by showing that neither problem can be solved in time $n^{o(\mathrm{td})}$, under the ETH, by employing a recursive low tree-depth construction that may be of independent interest.
- Finally, we consider a structural parameter that is known to be restrictive enough to render both problems FPT: vertex cover. For both problems the best known algorithm in this setting has a super-exponential dependence of the form $\mathrm{vc}^{\mathcal{O}(\mathrm{vc})}$. We show that this is optimal, as an algorithm with dependence of the form $\mathrm{vc}^{o(\mathrm{vc})}$ would violate the ETH. Our proof relies on a new application of the technique of $d$-detecting families introduced by Bonamy et al. [ToCT 2019].

Our results, although mostly negative in nature, paint a clear picture regarding the complexity of both problems in the landscape of parameterized complexity, since in all cases we provide essentially matching upper and lower bounds.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** ETH, Parameterized Complexity, SETH

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.77

**Related Version** *Full Version*: https://arxiv.org/abs/2304.14724

**Funding** This work is partially supported by ANR projects ANR-21-CE48-0022 (S-EX-AP-PE-AL) and ANR-18-CE40-0025-01 (ASSK).

## 1 Introduction

Parameterized complexity and in particular the study of structural parameters such as treewidth is one of the most well-developed approaches for dealing with NP-hard problems on graphs. Treewidth is of course one of the major success stories of this field, as a plethora of hard problems become fixed-parameter tractable when parameterized by this parameter.





Naturally, this success has motivated the effort to trace the limits of the algorithmic power of treewidth by attempting to understand what are the problems for which treewidth-based techniques *cannot* work.

When could the treewidth toolbox fail? One common scenario that seems to be shared by a multitude of problems which are W[1]-hard[1] parameterized by treewidth is when a natural dynamic programming algorithm does exist, but the DP is forced to store for each vertex of a bag in the tree decomposition an arbitrarily large integer – for example a number related to the degree of the vertex. Our goal in this paper is to study situations of this type and pose the natural question of whether one can do better than the "obvious" DP, by obtaining an algorithm with better running time, even at the expense of looking at a parameter more restrictive than treewidth.

Given the above, we focus on two problems which are arguably among the most natural representatives of our scenario: Bounded Degree Vertex Deletion and Defective Coloring. In both problems the input is a graph $G$ and a target degree $\Delta$ and we are asked, in the case of Bounded Degree Vertex Deletion to delete a minimum number of vertices so that the remaining graph has degree at most $\Delta$, and in the case of Defective Coloring to partition $G$ into a minimum number of color classes such that each class induces a graph of degree at most $\Delta$. Both problems are well-studied, as they generalize classical problems (Vertex Cover and Coloring respectively) and we review some of the previous work below. However, the most relevant aspect of the two problems for our purposes is the following: (i) both problems admit DP algorithms with complexity of the form $n^{\mathcal{O}(\mathrm{tw})}$ and (ii) both problems are W[1]-hard parameterized by treewidth; in fact, for Defective Coloring, it is even known that assuming the ETH it cannot be solved in time $n^{o(\mathrm{tw})}$ [5].

Since the $n^{\mathcal{O}(\mathrm{tw})}$ algorithms follow from standard DP techniques, it becomes a natural question whether we can do better. Does a better algorithm exist? Realistically, one could hope for one of two things: either an algorithm which still handles the problem parameterized by treewidth and in view of the aforementioned lower bound only attempts a fine-grained improvement in the running time; or an algorithm which is qualitatively faster at the expense of using a more restricted parameter. The results of this paper give strong negative evidence for both questions: if we parameterize by treewidth (and even by pathwidth) the running time of the standard DP is optimal under the SETH even for all fixed values of the other relevant parameters ($\Delta$ and the number of colors $\chi_{\mathrm{d}}$); while if we parameterize by more restrictive parameters, such as tree-depth and vertex cover, we obtain lower bound results (under the ETH) which indicate that the best algorithm is still essentially to run a form of the standard treewidth DP, even in these much more restricted cases. Our results thus paint a complete picture of the structurally parameterized complexity of these two problems and indicate that the standard DP is optimal in a multitude of restricted cases.

**Our contribution in more detail.** Following standard techniques, the two problems admit DP algorithms with tables of sizes $(\Delta+2)^{\mathrm{tw}}$ and $(\chi_{\mathrm{d}}(\Delta+1))^{\mathrm{tw}}$ respectively. Our first result is a collection of reductions proving that, assuming the SETH, no algorithm can improve upon these dynamic programs, even for pathwidth. More precisely, we show that no algorithm can solve Bounded Degree Vertex Deletion and Defective Coloring in time $(\Delta+2-\varepsilon)^{\mathrm{pw}}n^{\mathcal{O}(1)}$ and $(\chi_{\mathrm{d}}(\Delta+1)-\varepsilon)^{\mathrm{pw}}n^{\mathcal{O}(1)}$ respectively, for any $\varepsilon>0$ and for

[1] We assume the reader is familiar with the basics of parameterized complexity theory, as given in standard textbooks [14].

any combination of fixed values of $\Delta, \chi_d$ (except the combination $\Delta = 0$ and $\chi_d = 2$, which trivially makes Defective Coloring polynomial-time solvable). Our reductions follow the general strategy pioneered by Lokshtanov, Marx, and Saurabh [37] and indeed generalize their results for Vertex Cover and Coloring (which already covered the case $\Delta = 0$). The main difficulty here is being able to cover all values of the secondary parameters and for technical reasons we are forced to give separate versions of our reductions to cover the case $\Delta = 1$ for both problems. Along the way we note that, even though an algorithm with complexity $(\chi_d\Delta)^{\mathcal{O}(\mathrm{tw})}n^{\mathcal{O}(1)}$ was given for Defective Coloring in [5], it was not known if an algorithm with complexity $(\chi_d(\Delta+1))^{\mathrm{tw}}n^{\mathcal{O}(1)}$ (that is, with a quasi-linear dependence on the table size) is possible. For completeness, we settle this by providing an algorithm of this running time, using the FFT technique proposed by Cygan and Pilipczuk [17]. Taking also into account the Bounded Degree Vertex Deletion algorithm of running time $(\Delta+2)^{\mathrm{tw}}n^{\mathcal{O}(1)}$ given by van Rooij [48], we have exactly matching upper and lower bounds for both problems, for both treewidth and pathwidth.

Given that the results above show rather conclusively that the standard DP is the best algorithm for parameters treewidth and pathwidth, we then move on to a more restricted case: tree-depth. We recall that graphs of tree-depth $k$ are a proper subclass of graphs of pathwidth $k$, therefore one could reasonably hope to obtain a better algorithm for this parameter. This hope may further be supported by the fact that known lower bounds do not match the complexity of the standard algorithm. More precisely, the W[1]-hardness reduction given for Bounded Degree Vertex Deletion parameterized by tree-depth by Ganian, Klute, and Ordyniak [27] has a quartic blow-up, thus only implying that no $n^{o(\sqrt[4]{\mathrm{td}})}$ algorithm is possible; while the reduction given for Defective Coloring in [5] has a quadratic blow-up, only implying that no $n^{o(\sqrt{\mathrm{td}})}$ algorithm is possible (in both cases under the ETH). Our contribution is to show that both reductions can be replaced by more efficient reductions which are *linear* in the parameter; we thus establish that neither problem can be solved in time $n^{o(\mathrm{td})}$, implying that the treewidth-based algorithm remains (qualitatively) optimal even in this restricted case. One interesting aspect of our reductions is that, rather than using a modulator to a low tree-depth graph, which is common in such reductions, we use a recursive construction that leverages the full power of the parameter and may be of further use in tightening other lower bounds for the parameter tree-depth.

Finally, we move on to a more special case, parameterizing both problems by vertex cover. Both problems are FPT for this parameter and, since vertex cover is very restrictive as a parameter, one would hope that, finally, we should be able to obtain an algorithm that is more clever than the treewidth-based DP. Somewhat disappointingly, the known FPT algorithms for both problems have complexity $\mathrm{vc}^{\mathcal{O}(\mathrm{vc})}n^{\mathcal{O}(1)}$ [5], and the super-exponential dependence on the parameter is due to the fact that both algorithms are simple win/win arguments which, in one case, just execute the standard treewidth DP. We show that this is justified, as neither problem can be solved in time $\mathrm{vc}^{o(\mathrm{vc})}n^{\mathcal{O}(1)}$ (under the ETH), meaning that the algorithm that blindly executes the treewidth-based DP in some cases is still (qualitatively) best possible. We obtain our result by applying the technique of $d$-detecting families, introduced by Bonamy et al. [10]. Our results indicate that parameterization by vertex cover is a domain where this promising, but currently under-used, technique may find more applications in parameterized complexity.

**Related work.** Both Bounded Degree Vertex Deletion and Defective Coloring are well-studied problems with a rich literature. Bounded Degree Vertex Deletion finds application in a multitude of areas, ranging from computational biology [21] to some related problems in voting theory [7, 9], and its dual problem, called $s$-Plex Detection,

has numerous applications in social network analysis [4, 39, 40]. Various approximation algorithms are known [24, 25, 43]. The problem has also been extensively studied under the scope of parameterized complexity. It is W[2]-hard for unbounded values of $\Delta$ and parameter $k$ (the value of the optimal) [21], while it admits a linear-size kernel parameterized by $k$ [21, 50], for any fixed $\Delta \geq 0$; numerous FPT algorithms have been presented in the latter setting [40, 41, 49]. FPT approximation algorithms were given for Bounded Degree Vertex Deletion in [34] and [38]. As for Defective Coloring, which also appears in the literature as Improper Coloring, it was introduced almost 40 years ago [1, 13]. The main motivation behind this problem comes from the field of telecommunications, where the colors correspond to available frequencies and the goal is to assign them to communication nodes; a small amount of interference between neighboring nodes may be tolerable, which is modeled by the parameter $\Delta$. There have been plenty of works on the problem (see [2, 3, 5, 6, 12, 30] and the references therein), especially on unit disk graphs and various classes of grids.

The previous work for both problems that is most relevant to us focuses on their parameterized complexity for structural parameters, such as treewidth. In this setting, Bounded Degree Vertex Deletion was one of the first problems to be discovered to be W[1]-hard parameterized by treewidth [8], though the problem does become FPT parameterized by $\mathrm{tw} + \Delta$ or $\mathrm{tw} + k$. This hardness result was more recently improved by Ganian et al. [27], who showed that Bounded Degree Vertex Deletion is W[1]-hard parameterized by tree-depth and feedback vertex set. Defective Coloring was shown to be W[1]-hard parameterized by tree-depth (and hence pathwidth and treewidth) in [5]. However, [5] gave a hardness reduction for pathwidth that is linear in the parameter, and hence implies a $n^{o(\mathrm{pw})}$ lower bound for Defective Coloring under the ETH, but a hardness reduction for tree-depth that is quadratic (implying only a $n^{o(\sqrt{\mathrm{td}})}$ lower bound). Similarly, the reduction given by [27] for Bounded Degree Vertex Deletion parameterized by tree-depth is quartic in the parameter, as it goes through an intermediate problem (a variant of Subset Sum), implying only a $n^{o(\sqrt[4]{\mathrm{td}})}$ lower bound. Defective Coloring is known to be FPT parameterized by vertex cover using a simple win/win argument which applies the treewidth-based DP in one case (if $\Delta > \mathrm{vc}$, then the graph is always 2-colorable; otherwise the standard DP algorithm runs in FPT time), and the same is true for Bounded Degree Vertex Deletion (if $\Delta \leq \mathrm{vc}$, we can use the aforementioned FPT algorithm for parameters $\mathrm{tw} + \Delta$, else assume that $k < \mathrm{vc}$, as otherwise the problem is trivial, follow the reduction of [8] to Vector Dominating Set [44] and notice that at most vc vertices have degree greater than $\Delta$). Hence, the best algorithms for both problems for this parameter have complexity $\mathrm{vc}^{\mathcal{O}(\mathrm{vc})}n^{\mathcal{O}(1)}$.

The fine-grained analysis of the complexity of structural parameterizations, such as by treewidth, is an active field of research. The technique of using the SETH to establish tight running time lower bounds was pioneered by Lokshtanov, Marx, and Saurabh [37]. Since then, tight upper and lower bounds are known for a multitude of problems for parameterizations by treewidth and related parameters, such as pathwidth and clique-width [15, 16, 19, 20, 23, 26, 28, 29, 31, 42, 47]. One difficulty of the results we present here is that we need to present a family of reductions: one for each fixed value of $\Delta$ and $\chi_{\mathrm{d}}$. There are a few other problems for which families of tight lower bounds are known, such as $k$-Coloring, for which the correct dependence is $k^{\mathrm{tw}}$ for treewidth [37] and $(2^k - 2)^{\mathrm{cw}}$ for clique-width [35] for all $k \geq 3$; distance $r$-Dominating Set, for which the correct dependence is $(2r+1)^{\mathrm{tw}}$ [11] and $(3r+1)^{\mathrm{cw}}$ [32], for all $r \geq 1$; and distance $d$-Independent Set, for which the correct dependence is $d^{\mathrm{tw}}$ [33]. In all these cases, the optimal algorithm is the "natural" DP, and our results for Bounded Degree Vertex Deletion and Defective Coloring fit this pattern.

**Table 1** Lower bounds established in the current work. The results of the first row are under SETH, while all the rest under ETH.

| Parameter | Bounded Degree Vertex Deletion | Defective Coloring |
|---|---|---|
| pathwidth + $\Delta$ | $\mathcal{O}^\star((\Delta+2-\varepsilon)^{\mathrm{pw}})$ | $\mathcal{O}^\star((\chi_{\mathrm{d}}\cdot(\Delta+1)-\varepsilon)^{\mathrm{pw}})$ |
| treedepth | $n^{o(\mathrm{td})}$ | $n^{o(\mathrm{td})}$ |
| vertex cover | $\mathrm{vc}^{o(\mathrm{vc})}n^{\mathcal{O}(1)}$ | $\mathrm{vc}^{o(\mathrm{vc})}n^{\mathcal{O}(1)}$ |

Even though the previous work mentioned above may make it seem that our SETH-based lower bounds are not surprising, it is important to stress that it is not a given that the naïve DP should be optimal for our problems. In particular, Bounded Degree Vertex Deletion falls into a general category of $(\sigma,\rho)$-domination problems, which were studied recently in [22] (we refer the reader there for the definition of $(\sigma,\rho)$-domination). One of the main results of that work was to show that significant improvements over the basic DP are indeed possible in some cases, and in particular when one of $\sigma,\rho$ is cofinite. Since Bounded Degree Vertex Deletion is the case where $\sigma=\{0,\ldots,\Delta\}$ and $\rho=\mathbb{N}$ (that is, $\rho$ is co-finite), our result falls exactly in the territory left uncharted by [22], where more efficient algorithms could still be found (and where indeed [22] did uncover such algorithms for some values of $\sigma,\rho$).

**Organization.** In Section 2 we discuss the general preliminaries, followed by the lower bounds for Bounded Degree Vertex Deletion in Sections 3–5, and the conclusion in Section 6. Proofs marked with $(\star)$, as well as all the results pertaining to Defective Coloring, can be found in the full version of the paper.

## 2 Preliminaries

Throughout the paper we use standard graph notation [18], and we assume familiarity with the basic notions of parameterized complexity [14]. All graphs considered are undirected without loops, unless explicitly stated otherwise. For a graph $G=(V,E)$ and two integers $\chi_{\mathrm{d}}\geq 1,\Delta\geq 0$, we say that $G$ admits a $(\chi_{\mathrm{d}},\Delta)$-coloring if one can partition $V$ into $\chi_{\mathrm{d}}$ sets such that the graph induced by each set has maximum degree at most $\Delta$. In that case, Defective Coloring is the problem of deciding, given $G,\chi_{\mathrm{d}},\Delta$, whether $G$ admits a $(\chi_{\mathrm{d}},\Delta)$-coloring. For $x,y\in\mathbb{Z}$, let $[x,y]=\{z\in\mathbb{Z}\mid x\leq z\leq y\}$, while $[x]=[1,x]$. Standard $\mathcal{O}^\star$ notation is used to suppress polynomial factors. For function $f:A\to B$, and $A'\subseteq A$, let $f(A')=\sum_{a\in A'}f(a)$. For the pathwidth bounds, we use the notion of *mixed search strategy* [45], where an edge is cleared by either placing a searcher on both of its endpoints or sliding one along the edge. We rely on a weaker form of the ETH, which states that 3-SAT on instances with $n$ variables and $m$ clauses cannot be solved in time $2^{o(n+m)}$.

In $k$-Multicolored Clique, we are given a graph $G=(V,E)$ and a partition of $V$ into $k$ independent sets $V_1,\ldots,V_k$, each of size $n$, and we are asked to determine whether $G$ contains a $k$-clique. It is well-known that this problem does not admit any $f(k)n^{o(k)}$ algorithm, where $f$ is any computable function, unless the ETH is false [14].

In $q$-CSP-$B$, we are given a Constraint Satisfaction (CSP) instance with $n$ variables and $m$ constraints. The variables take values in a set $Y$ of size $B$, i.e. $|Y|=B$. Each constraint involves at most $q$ variables and is given as a list of satisfying assignments for these variables, where a satisfying assignment is a $q$-tuple of values from the set $Y$ given to each of the $q$ variables. The following result was shown by Lampis [35] to be a natural consequence of the SETH, and has been used in the past for various hardness results [19, 20, 29].

▶ **Theorem 1** ([35]). *For any $B \geq 2$ it holds that, if the SETH is true, then for all $\varepsilon > 0$, there exists a $q$ such that $n$-variable $q$-CSP-$B$ cannot be solved in time $\mathcal{O}^\star((B-\varepsilon)^n)$.*

## 3 Treewidth and Maximum Degree Lower Bounds

In this section we present tight lower bounds on the complexity of solving both Bounded Degree Vertex Deletion and Defective Coloring parameterized by the treewidth of the input graph plus the target degree. The latter result can be found in the full version of the paper.

Both reductions are similar in nature: we start from an instance $\phi$ of $q$-CSP-$B$, and produce an equivalent instance on a graph of pathwidth $\mathrm{pw} = n + \mathcal{O}(1)$, where $n$ denotes the number of variables of $\phi$. An interesting observation however, is that for both problems, we have to distinguish between the case where $\Delta = 1$ and $\Delta \geq 2$; the whole construction becomes much more complicated in the second case.

### 3.1 Bounded Degree Vertex Deletion

In the following, we will present a reduction from $q$-CSP-$B$ to Bounded Degree Vertex Deletion, for any fixed $\Delta \geq 1$, where $\Delta = B - 2$. In that case, if there exists a $\mathcal{O}^\star((\Delta + 2 - \varepsilon)^{\mathrm{pw}})$ algorithm for Bounded Degree Vertex Deletion, where $\varepsilon > 0$, then there exists a $\mathcal{O}^\star((B-\varepsilon)^n)$ algorithm for $q$-CSP-$B$, for any constant $q$, which due to Theorem 1 results in SETH failing.

Our reduction is based on the construction of "long paths" of *Block gadgets*, that are serially connected in a path-like manner. Each such "path" corresponds to a variable of the given formula, while each column of this construction is associated with one of its constraints. Intuitively, our aim is to embed the $B^n$ possible variable assignments into the $(\Delta+2)^{\mathrm{tw}}$ states of some optimal dynamic program that would solve the problem on our constructed instance.

Below, we present a sequence of gadgets used in our reduction. The aforementioned block gadgets, which allow a solution to choose among $\Delta + 2$ reasonable choices, are the main ingredient. Notice that these gadgets will differ significantly depending on whether $\Delta$ is equal to 1 or not. We connect these gadgets in a path-like manner that ensures that choices remain consistent throughout the construction, and connect constraint gadgets in different "columns" of the constructed grid in a way that allows us to verify if the choice made represents a satisfying assignment, without increasing the graph's treewidth.

▶ **Theorem 2.** *For any constant $\varepsilon > 0$, there is no $\mathcal{O}^\star((3-\varepsilon)^{\mathrm{pw}})$ algorithm deciding Bounded Degree Vertex Deletion for $\Delta = 1$, where* pw *denotes the input graph's pathwidth, unless the SETH is false.*

**Proof.** Fix some positive $\varepsilon > 0$ for which we want to prove the theorem. We will reduce $q$-CSP-3, for some $q$ that is a constant that only depends on $\varepsilon$, to Bounded Degree Vertex Deletion for $\Delta = 1$ in a way that ensures that if the resulting Bounded Degree Vertex Deletion instance could be solved in time $\mathcal{O}^\star((3-\varepsilon)^{\mathrm{pw}})$, then we would obtain an algorithm for $q$-CSP-3 that would contradict the SETH. To this end, let $\phi$ be an instance of $q$-CSP-3 of $n$ variables $X = \{x_i \mid i \in [n]\}$ taking values over the set $Y = [3]$ and $m$ constraints $C = \{c_j \mid j \in [m]\}$. For each constraint we are given a set of at most $q$ variables which are involved in this constraint and a list of satisfying assignments for these variables, the size of which is denoted by $s : C \to [3^q]$, i.e. $s(c_j) \leq 3^q = \mathcal{O}(1)$ denotes the number of satisfying assignments for constraint $c_j$. We will construct in polynomial time an equivalent instance $\mathcal{I} = (G, k)$ of Bounded Degree Vertex Deletion for $\Delta = 1$, where $\mathrm{pw}(G) \leq n + \mathcal{O}(1)$.

**Block and Variable Gadgets.** For every variable $x_i$ and every constraint $c_j$, construct a path of 3 vertices $p^1_{i,j}, p^2_{i,j}$ and $p^3_{i,j}$, which comprises the *block gadget* $\hat{B}_{i,j}$. Intuitively, we will map the deletion of $p^y_{i,j}$ with an assignment where $x_i$ receives value $y$. Next, for $j \in [m-1]$, we add an edge between $p^3_{i,j}$ and $p^1_{i,j+1}$, thus resulting in $n$ paths $P_1, \ldots, P_n$ of length $3m$, called *variable gadgets*.

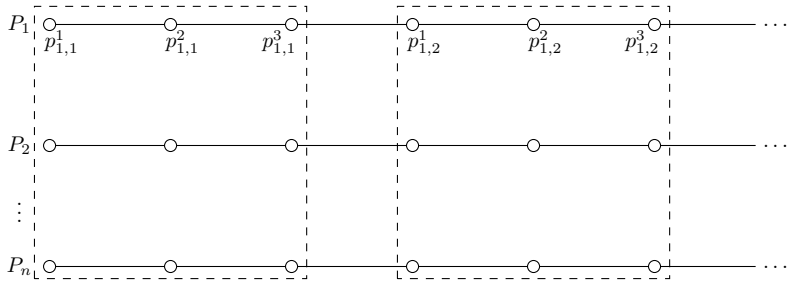



**Figure 1** Sequences of block gadgets comprise the variable gadgets.

**Constraint Gadget.** This gadget is responsible for determining constraint satisfaction, based on the choices made in the rest of the graph. For constraint $c_j$, construct the *constraint gadget* $\hat{C}_j$ as follows:

- construct a clique of $s(c_j)$ vertices $v^j_1, \ldots, v^j_{s(c_j)}$, and fix an arbitrary one-to-one mapping between those vertices and the satisfying assignments of $c_j$,
- attach to each vertex $v^j_\ell$ a leaf $l^j_\ell$,
- if variable $x_i$ is involved in the constraint $c_j$ and $v^j_\ell$ corresponds to an assignment where $x_i$ has value $y \in Y$, add an edge between $v^j_\ell$ and $p^y_{i,j}$.

Let graph $G_0$ be the graph containing all variable gadgets $P_i$ as well as all the constraint gadgets $\hat{C}_j$, for $i \in [n]$ and $j \in [m]$. To construct graph $G$, introduce $\kappa = 2n+1$ copies $G_1, \ldots, G_\kappa$ of $G_0$, such that they are connected sequentially as follows: for $i \in [n]$ and $j \in [\kappa - 1]$, add an edge between $p^3_{i,m}(G_j)$ and $p^1_{i,1}(G_{j+1})$, where $p^y_{i,j}(G_a)$ denotes the vertex $p^y_{i,j}$ of graph $G_a$. We refer to the block gadget $\hat{B}_{i,j}$, to the variable gadget $P_i$ and to the constraint gadget $\hat{C}_j$ of $G_a$ as $\hat{B}^{G_a}_{i,j}$, $P^{G_a}_i$ and $\hat{C}^{G_a}_j$ respectively. Let $\mathcal{P}^i$ denote the path resulting from $P^{G_1}_i, \ldots, P^{G_\kappa}_i$. Let $k' = \sum_{j=1}^{m}(s(c_j) - 1 + n) = m \cdot n + \sum_{j=1}^{m}(s(c_j) - 1)$, and set $k = \kappa \cdot k'$.

▶ **Lemma 3.** ($\star$) *If $\phi$ is satisfiable, then there exists $S \subseteq V(G)$ such that $G - S$ has maximum degree at most 1 and $|S| \leq k$.*

▶ **Lemma 4.** ($\star$) *If there exists $S \subseteq V(G)$ such that $G - S$ has maximum degree at most 1 and $|S| \leq k$, then $\phi$ is satisfiable.*

▶ **Lemma 5.** ($\star$) *It holds that $\mathrm{pw}(G) \leq n + \mathcal{O}(1)$.*

Therefore, in polynomial time, we can construct a graph $G$, of pathwidth $\mathrm{pw}(G) \leq n + \mathcal{O}(1)$ due to Lemma 5, such that, due to Lemmas 3 and 4, deciding whether there exists $S \subseteq V(G)$ of size $|S| \leq k$ and $G - S$ has maximum degree at most 1 is equivalent to deciding whether $\phi$ is satisfiable. In that case, assuming there exists a $\mathcal{O}^\star((3-\varepsilon)^{\mathrm{pw}(G)})$ algorithm for Bounded Degree Vertex Deletion for $\Delta = 1$, one could decide $q$-CSP-3 in time $\mathcal{O}^\star((3-\varepsilon)^{\mathrm{pw}(G)}) = \mathcal{O}^\star((3-\varepsilon)^{n+\mathcal{O}(1)}) = \mathcal{O}^\star((3-\varepsilon)^n)$ for any constant $q$, which contradicts the SETH due to Theorem 1. ◀

▶ **Theorem 6.** ($\star$) *For any constant $\varepsilon > 0$, there is no $\mathcal{O}^\star((\Delta + 2 - \varepsilon)^{\mathrm{pw}})$ algorithm deciding* Bounded Degree Vertex Deletion *for $\Delta \geq 2$, where* pw *denotes the input graph's pathwidth, unless the SETH is false.*

## 4 Tree-depth Lower Bounds

In this section we present lower bounds on the complexity of solving Bounded Degree Vertex Deletion and Defective Coloring, when parameterized by the tree-depth of the input graph. The latter result can be found in the full version of the paper.

The common starting point of both reductions is an instance of $k$-Multicolored Clique, where $k$ is a power of 2. Our main technical contribution is a recursive construction which allows us to keep the tree-depth of the constructed graph linear with respect to $k$, which we briefly sketch here. The main idea behind the construction is the following:

- First, we describe a *choice gadget* $\hat{C}_i$, which encodes, for every independent set $V_i$ of the graph, which vertex of $V_i$ is part of the clique.
- Afterwards, we describe how one can make a *copy* of such a choice gadget, by using only a constant number of vertices, while at the same time guaranteeing that the choices between the instances of the choice gadget remain the same.
- Lastly, we define an *adjacency gadget* $\hat{A}(i_1, i_2, i'_1, i'_2)$, whose purpose is to verify that, for the given choices of vertices, there exists an edge between $V_i$ and $V_{i'}$, for any $i \in [i_1, i_2]$ and $i' \in [i'_1, i'_2]$. Initially we deal with the case where $i_1 = i_2$ and $i'_1 = i'_2$, while ensuring that the tree-depth of the construction is constant. For the other case, the gadget is constructed in two steps. Firstly, it contains all the choice gadgets $\hat{C}_i$ and $\hat{C}_{i'}$. Secondly, it contains 4 instances of adjacency gadgets, due to the upper and lower half of $[i_1, i_2]$ and $[i'_1, i'_2]$, while the occurrences of the choice gadgets in those are copies of the choice gadgets introduced in the first step. The fact that $k$ is a power of 2 guarantees that the upper and lower half of both $[i_1, i_2]$ and $[i'_1, i'_2]$ are well defined.

Then, by removing the vertices used in the copy gadgets, it follows that all adjacency gadgets constructed in the second step become disconnected. Therefore, the tree-depth of the whole construction is given by a recursive formula of the form $T(k) = \mathcal{O}(k) + T(k/2)$.

### 4.1 Bounded Degree Vertex Deletion

▶ **Theorem 7.** *For any computable function $f$, if there exists an algorithm that solves* Bounded Degree Vertex Deletion *in time $f(\mathrm{td})n^{o(\mathrm{td})}$, where* td *denotes the tree-depth of the input graph, then the ETH is false.*

**Proof.** Let $(G, k)$ be an instance of $k$-Multicolored Clique, such that every vertex of $G$ has a self loop, i.e. $\{v, v\} \in E(G)$, for all $v \in V(G)$. Recall that we assume that $G$ is given to us partitioned into $k$ independent sets $V_1, \ldots, V_k$, where $V_i = \{v_1^i, \ldots, v_n^i\}$. Assume without loss of generality that $k = 2^z$, for some $z \in \mathbb{N}$ (one can do so by adding dummy independent sets connected to all the other vertices of the graph). Moreover, let $E^{i_1,i_2} \subseteq E(G)$ denote the edges of $G$ with one endpoint in $V_{i_1}$ and the other in $V_{i_2}$. Set $\Delta = n^3$. We will construct in polynomial time a graph $H$ of tree-depth $\mathrm{td}(H) = \mathcal{O}(k)$ and size $|V(H)| = k^{\mathcal{O}(1)} \cdot n^{\mathcal{O}(1)}$, such that there exists $S \subseteq V(H)$, $|S| \leq k'$ and $H - S$ has maximum degree at most $\Delta$, for some $k'$, if and only if $G$ has a $k$-clique.

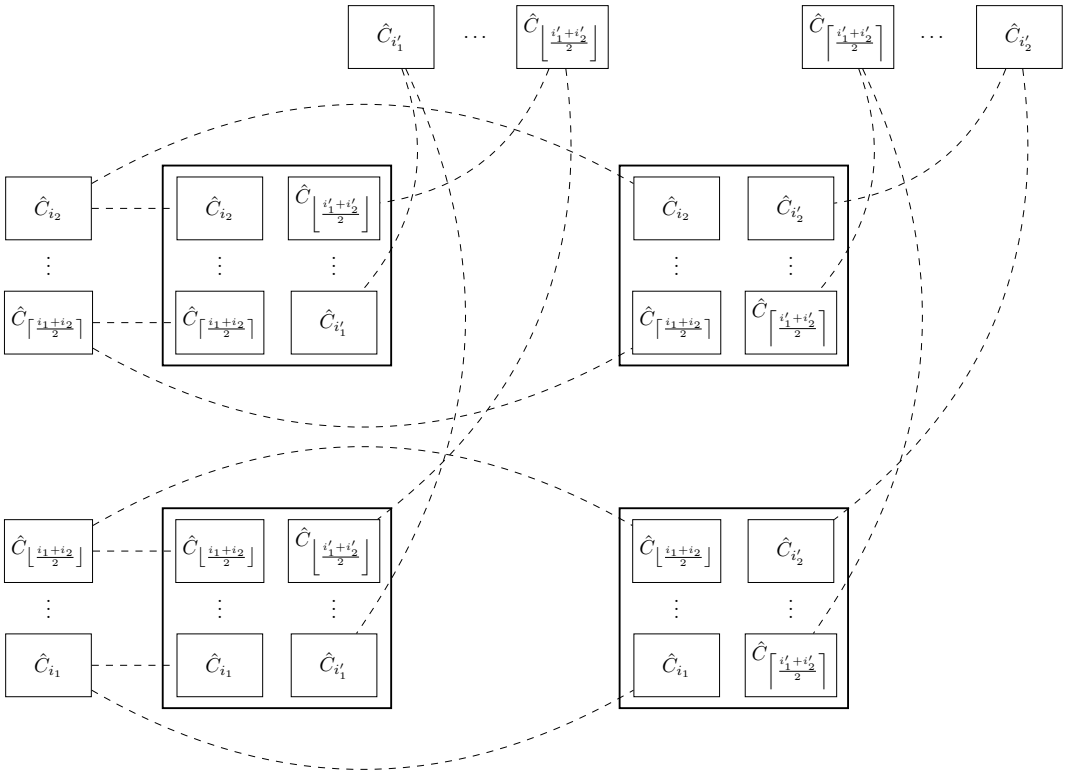


**Figure 2** Adjacency gadget $\hat{A}(i_1, i_2, i'_1, i'_2)$. Dashed lines denote copies.

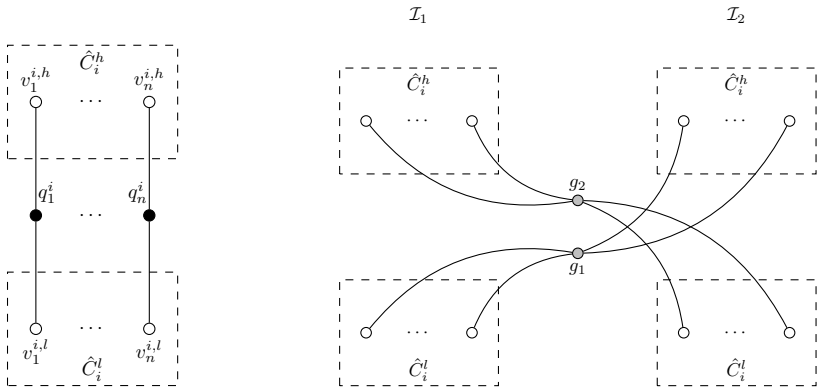
**Choice Gadget.** For an independent set $V_i$, we construct the *choice gadget* $\hat{C}_i$ as depicted in Figure 3a. We first construct independent sets $\hat{C}_i^p = \{v_1^{i,p}, \ldots, v_n^{i,p}\}$, where $p \in \{h, l\}$. Afterwards, we connect $v_j^{i,h}$ and $v_j^{i,l}$ with a vertex $q_j^i$, and add to the latter $\Delta - 1$ leaves. Intuitively, we will consider an one-to-one mapping between the vertex $v_j^i$ of $V_i$ belonging to a supposed $k$-clique of $G$ and the deletion of exactly $j$ vertices of $\hat{C}_i^l$ and $n - j$ from $\hat{C}_i^h$.

**Copy Gadget.** Given two instances $\mathcal{I}_1$, $\mathcal{I}_2$ of a choice gadget $\hat{C}_i$, when we say that we connect them with a *copy gadget*, we introduce two vertices $g_1$ and $g_2$, attach to each of those $\Delta - n$ leaves, and lastly add an edge between $g_1$ (respectively, $g_2$) with the vertices of $\hat{C}_i^l$ of instance $\mathcal{I}_1$ (respectively, $\mathcal{I}_2$), as well as the vertices of $\hat{C}_i^h$ of instance $\mathcal{I}_2$ (respectively, $\mathcal{I}_1$).

**Edge Gadget.** Let $e = \{v_{j_1}^{i_1}, v_{j_2}^{i_2}\} \in E^{i_1,i_2}$ be an edge of $G$. Construct the *edge gadget* $\hat{E}_e$ as depicted in Figure 4, where every vertex $c_j^i$ has $\Delta$ leaves attached.

**Adjacency Gadget.** For $i_1 \leq i_2$ and $i'_1 \leq i'_2$, we define the *adjacency gadget* $\hat{A}(i_1, i_2, i'_1, i'_2)$ as follows:

- Consider first the case when $i_1 = i_2$ and $i'_1 = i'_2$. Let the adjacency gadget contain instances of the edge gadgets $\hat{E}_e$, for $e \in E^{i_1,i'_1}$, the choice gadgets $\hat{C}_{i_1}$ and $\hat{C}_{i'_1}$, as well as vertices $\ell^l_{i_1,i'_1}$, $\ell^h_{i_1,i'_1}$, $r^l_{i_1,i'_1}$ and $r^h_{i_1,i'_1}$. Add edges between
  - $\ell^l_{i_1,i'_1}$ and $\hat{C}^l_{i_1}$,
  - $\ell^h_{i_1,i'_1}$ and $\hat{C}^h_{i_1}$,
  - $r^l_{i_1,i'_1}$ and $\hat{C}^l_{i'_1}$,
  - $r^h_{i_1,i'_1}$ and $\hat{C}^h_{i'_1}$.




**(a)** Choice gadget $\hat{C}_i$. **(b)** Making a copy of a choice gadget $\hat{C}_i$.

**Figure 3** Black vertices have $\Delta - 1$ and gray $\Delta - n$ leaves attached.

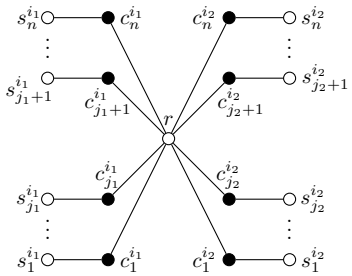



**Figure 4** Edge gadget $\hat{E}_e$ for $e = \{v_{j_1}^{i_1}, v_{j_2}^{i_2}\}$. Black vertices have $\Delta$ leaves attached.

If $e = \{v_{j_1}^{i_1}, v_{j_2}^{i'_1}\} \in E^{i_1,i'_1}$, then add the following edges adjacent to $\hat{E}_e$:

- $\ell^l_{i_1,i'_1}$ with $s^{i_1}_\kappa$, for $\kappa \in [j_1]$,
- $\ell^h_{i_1,i'_1}$ with $s^{i_1}_\kappa$, for $\kappa \in [j_1+1, n]$,
- $r^l_{i_1,i'_1}$ with $s^{i'_1}_\kappa$, for $\kappa \in [j_2]$,
- $r^h_{i_1,i'_1}$ with $s^{i'_1}_\kappa$, for $\kappa \in [j_2+1, n]$.

Let $\tau(x)$, where $x \in \{\ell^l_{i_1,i'_1}, \ell^h_{i_1,i'_1}, r^l_{i_1,i'_1}, r^h_{i_1,i'_1}\}$, denote the number of neighbors of $x$ belonging to some edge gadget. Attach $\Delta - \tau(x)$ leaves to vertex $x$.

- Now consider the case when $i_1 < i_2$ and $i'_1 < i'_2$. Then, let $\hat{A}(i_1, i_2, i'_1, i'_2)$ contain choice gadgets $\hat{C}_i$ and $\hat{C}_{i'}$, where $i \in [i_1, i_2]$ and $i' \in [i'_1, i'_2]$, which we will refer to as the *original choice gadgets* of $\hat{A}(i_1, i_2, i'_1, i'_2)$, as well as the adjacency gadgets

- $\hat{A}(i_1, \lfloor \frac{i_1+i_2}{2} \rfloor, i'_1, \lfloor \frac{i'_1+i'_2}{2} \rfloor)$,
- $\hat{A}(i_1, \lfloor \frac{i_1+i_2}{2} \rfloor, \lceil \frac{i'_1+i'_2}{2} \rceil, i'_2)$,
- $\hat{A}(\lceil \frac{i_1+i_2}{2} \rceil, i_2, i'_1, \lfloor \frac{i'_1+i'_2}{2} \rfloor)$,
- $\hat{A}(\lceil \frac{i_1+i_2}{2} \rceil, i_2, \lceil \frac{i'_1+i'_2}{2} \rceil, i'_2)$.

Lastly, we connect with a copy gadget any choice gadgets $\hat{C}_i$ and $\hat{C}_{i'}$ appearing in said adjacency gadgets, with the corresponding original choice gadget $\hat{C}_i$ and $\hat{C}_{i'}$. Notice that then, every original choice gadget is taking part in two copy gadgets.

Let graph $H$ be the adjacency gadget $\hat{A}(1, k, 1, k)$. Notice that it holds that $|V(H)| = (n \cdot k)^{\mathcal{O}(1)}$. Let $\beta = 2k(2k-1)$, and set $k' = 2(|E(G)| - kn) \cdot 2n + kn \cdot 2n + 2\binom{k}{2} + k + n \cdot \beta$.

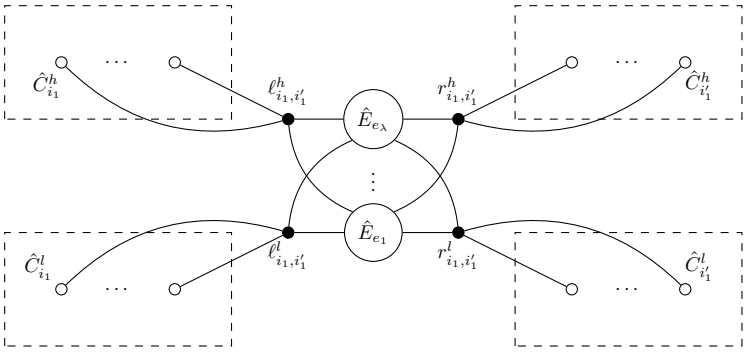



**Figure 5** Adjacency gadget $\hat{A}(i_1, i_1, i'_1, i'_1)$, where $E^{i_1,i'_1} = \{e_i \mid i \in [\lambda]\}$. Black vertices have leaves attached.

▶ **Lemma 8.** ($\star$) *$H$ has the following properties:*

- *The number of instances of choice gadgets present in $H$ is $\beta$,*
- *The number of instances of edge gadget $\hat{E}_e$ present in $H$, where $e = \{v^{i_1}_{j_1}, v^{i_2}_{j_2}\} \in E(G)$, is one if $i_1 = i_2$, and two otherwise.*

▶ **Lemma 9.** ($\star$) *It holds that* $\mathrm{td}(H) = \mathcal{O}(k)$.

▶ **Lemma 10.** ($\star$) *If $G$ contains a $k$-clique, then there exists $S \subseteq V(H)$, with $|S| \leq k'$, such that $H - S$ has maximum degree at most $\Delta$.*

▶ **Lemma 11.** ($\star$) *If there exists $S \subseteq V(H)$, with $|S| \leq k'$, such that $H - S$ has maximum degree at most $\Delta$, then $G$ contains a $k$-clique.*

Therefore, in polynomial time, we can construct a graph $H$, of tree-depth $\mathrm{td} = \mathcal{O}(k)$ due to Lemma 9, such that, due to Lemmas 10 and 11, deciding whether there exists $S \subseteq V(H)$ of size $|S| \leq k'$ and $H - S$ has maximum degree at most $\Delta = n^3$ is equivalent to deciding whether $G$ has a $k$-clique. In that case, assuming there exists a $f(\mathrm{td})|V(H)|^{o(\mathrm{td})}$ algorithm for Bounded Degree Vertex Deletion, where $f$ is any computable function, one could decide $k$-Multicolored Clique in time $f(\mathrm{td})|V(H)|^{o(\mathrm{td})} = g(k) \cdot n^{o(k)}$, for some computable function $g$, which contradicts the ETH. ◀

## 5 Vertex Cover Lower Bounds

In this section we present lower bounds on the complexity of solving Bounded Degree Vertex Deletion when parameterized by the vertex cover of the input graph. An analogous lower bound is shown for Defective Coloring, which has been deferred to the appendix due to space restrictions. In both cases, we start from a 3-SAT instance of $n$ variables, and produce an equivalent instance where the input graph has vertex cover $\mathcal{O}(n/\log n)$, hence any algorithm solving the latter problem in time $\mathrm{vc}^{o(\mathrm{vc})}n^{\mathcal{O}(1)}$ would refute the ETH. As a consequence of the above, already known algorithms for both of these problems are essentially optimal. We start by presenting some necessary tools used in both of these reductions, and then prove the stated results for Bounded Degree Vertex Deletion.

### 5.1 Preliminary Tools

We first define a constrained version of 3-SAT, called (3,4)-XSAT. This variant is closely related with the (3,4)-SAT problem [46] which asks whether a given formula $\phi$ is satisfiable, where $\phi$ is a 3-SAT formula each clause of which contains exactly 3 different variables and

each variable occurs in at most 4 clauses. As observed by Bonamy et al. [10], a corollary of Tovey's work [46] is that there is no $2^{o(n)}$ algorithm for (3,4)-SAT unless the ETH is false, where $n$ denotes the number of variables of the formula. Here we prove an analogous lower bound for (3,4)-XSAT. Subsequently, by closely following Lemma 3.2 from [10], we present a way to partition the formula's variables and clauses into groups such that variables appearing in clauses of the same clause group belong to different variable groups.

(3,4)-XSAT
**Input:** A 3-SAT formula $\phi$ every clause of which contains exactly 3 distinct variables and each variable appears in at most 4 clauses.
**Task:** Determine whether there exists an assignment to the variables of $\phi$ such that each clause has exactly one True literal.

▶ **Theorem 12.** ($\star$) *(3,4)-XSAT cannot be decided in time $2^{o(n)}$, where $n$ denotes the number of variables of the input formula, unless the ETH fails.*

We proceed by proving that, given a (3,4)-XSAT instance, we can partition the variables and clauses of the formula into groups such that variables appearing in clauses of the same clause group belong to different variable groups.

▶ **Lemma 13.** ($\star$) *Let $\phi$ be an instance of (3,4)-XSAT, where $V$ denotes the set of its $n$ variables and $C$ the set of its clauses. Moreover, let $b \leq \sqrt{n}$. One can produce in time $n^{\mathcal{O}(1)}$ a partition of $\phi$'s variables into $n_V$ disjoint sets $V_1, \ldots, V_{n_V}$ of size at most $b$ as well as a partition of its clauses into $n_C$ disjoint sets $C_1, \ldots, C_{n_C}$ of size at most $\sqrt{n}$, for some integers $n_V = \mathcal{O}(n/b)$ and $n_C = \mathcal{O}(\sqrt{n})$, such that, for any $i \in [n_C]$, any two variables appearing in clauses of $C_i$ belong to different variable subsets.*

▶ **Definition 14.** *A $d$-detecting family is a set of subsets of a finite set $U$ that can be used to distinguish between different functions $f, g : U \to \{0, \ldots, d-1\}$. Therefore, if $f \neq g$, there exists $U' \subseteq U$ such that $f(U') \neq g(U')$ and $U'$ belongs to said family.*

Lindström [36] has provided a deterministic construction of sublinear, $d$-detecting families, while Bonamy et al. [10] were the first to use them in the context of computational complexity, proving tight lower bounds for the Multicoloring problem under the ETH. The following theorem will be crucial towards proving the stated lower bounds.

▶ **Theorem 15** ([36]). *For every constant $d \in \mathbb{N}$ and finite set $U$, there is a $d$-detecting family $\mathcal{F}$ on $U$ of size $\frac{2|U|}{\log_d |U|} \cdot (1 + o(1))$. Moreover, $\mathcal{F}$ can be constructed in time polynomial in $|U|$.*

## 5.2 Bounded Degree Vertex Deletion

Let $\phi$ be an instance of (3,4)-XSAT of $n$ variables. Assume without loss of generality that $n$ is a power of 4 (this can be achieved by adding dummy variables to the instance if needed). Making use of Lemma 13, one can obtain in time $n^{\mathcal{O}(1)}$ the following:

- a partition of $\phi$'s variables into subsets $V_1, \ldots, V_{n_V}$, where $|V_i| \leq \log n$ and $n_V = \mathcal{O}(n/\log n)$,
- a partition of $\phi$'s clauses into subsets $C_1, \ldots, C_{n_C}$, where $|C_i| \leq \sqrt{n}$ and $n_C = \mathcal{O}(\sqrt{n})$,

where any two variables occurring in clauses of the same clause subset belong to different variable subsets. For $i \in [n_C]$, let $\{C_{i,1}, \ldots, C_{i,n^i_{\mathcal{F}}}\}$ be a 4-detecting family of subsets of $C_i$ for some $n^i_{\mathcal{F}} = \mathcal{O}(\sqrt{n}/\log n)$, produced in time $n^{\mathcal{O}(1)}$ due to Theorem 15. Moreover, let $n_{\mathcal{F}} = \max_{i=1}^{n_C} n^i_{\mathcal{F}}$. Define $\Delta = n^3$ and $k = n_V$. We will construct a graph $G = (V, E)$ such that there exists $S \subseteq V(G)$ of size $|S| \leq k$ and $G - S$ has maximum degree at most $\Delta$ if and only if there exists an assignment such that every clause of $\phi$ has exactly one True literal.

**Choice Gadget.** For each variable subset $V_i$, we define the choice gadget graph $G_i$ as follows:
- introduce vertices $\kappa_i$, $\lambda_i$ and $v_i^j$, where $j \in [n]$,
- add edges $\{\kappa_i, v_i^j\}$ and $\{\lambda_i, v_i^j\}$, for all $j \in [n]$,
- attach sufficiently many leaves to $\kappa_i$ and $\lambda_i$ such that their degree is $\Delta + 1$.

Let $\mathcal{V}_i = \{v_i^j \mid j \in [n]\}$, for $i = 1, \ldots, n_V$. We fix an arbitrary one-to-one mapping so that every vertex of $\mathcal{V}_i$ corresponds to a different assignment for the variables of $V_i$. Since $2^{|V_i|} \leq n$, there are sufficiently many vertices to uniquely encode all the different assignments of $V_i$. Let $\mathcal{V} = \mathcal{V}_1 \cup \ldots \cup \mathcal{V}_{n_V}$ denote the set of all such vertices.

**Clause Gadget.** For $i \in [n_C]$, let $C_i$ be a clause subset and $\{C_{i,1}, \ldots, C_{i,n_{\mathcal{F}}^i}\}$ its 4-detecting family. For every subset $C_{i,j}$ of the 4-detecting family, introduce vertices $c_{i,j}$ and $c'_{i,j}$. Add an edge between $c_{i,j}$ and $v_p^q$ if there exists variable $x \in V_p$ such that $x$ occurs in some clause $c \in C_{i,j}$, and $v_p^q$ corresponds to an assignment of $V_p$ that satisfies $c$. Due to Lemma 13, $c_{i,j}$ has exactly $|C_{i,j}| \cdot \frac{3n}{2}$ such edges: there are exactly $3|C_{i,j}|$ different variables appearing in clauses of $C_{i,j}$, each belonging to a different variable subset, and for each such variable, half the assignments of the corresponding variable subset result in the satisfaction of the corresponding clause of $C_{i,j}$. Attach to $c_{i,j}$ a sufficient number of leaves such that its total degree is $\Delta + |C_{i,j}|$. Moreover, for $v \in \mathcal{V}$, let $v \in N(c'_{i,j})$ if $v \notin N(c_{i,j})$. Notice that then, it holds that $N(c_{i,j}) \cup N(c'_{i,j}) \supseteq \mathcal{V}$, while $N(c_{i,j}) \cap N(c'_{i,j}) = \emptyset$. Lastly, attach to $c'_{i,j}$ a sufficient number of leaves such that its total degree is $\Delta + (k - |C_{i,j}|)$.

Let $\mathcal{I} = (G, \Delta, k)$ be an instance of Bounded Degree Vertex Deletion.

▶ **Lemma 16.** ($\star$) *It holds that* $\mathrm{vc}(G) = \mathcal{O}(n/\log n)$.

▶ **Lemma 17.** ($\star$) *If $\phi$ is a Yes instance of (3,4)-XSAT, then $\mathcal{I}$ is a Yes instance of Bounded Degree Vertex Deletion.*

▶ **Lemma 18.** ($\star$) *If $\mathcal{I}$ is a Yes instance of Bounded Degree Vertex Deletion, then $\phi$ is a Yes instance of (3,4)-XSAT.*

We can now prove the main theorem of this section.

▶ **Theorem 19.** ($\star$) *There is no* $\mathrm{vc}^{o(\mathrm{vc})} n^{\mathcal{O}(1)}$ *time algorithm for Bounded Degree Vertex Deletion, where* vc *denotes the size of the minimum vertex cover of the input graph, unless the ETH fails.*

## 6 Conclusion

In this work, we have examined in depth the complexity of Bounded Degree Vertex Deletion and Defective Coloring under the perspective of parameterized complexity. In particular, we have precisely determined the complexity of both problems parameterized by some of the most commonly used structural parameters. As a direction for future research, we consider the question of whether we could obtain a $n^{o(\mathrm{fvs})}$ lower bound for Bounded Degree Vertex Deletion as well as for Defective Coloring when $\chi_d = 2$, where fvs denotes the size of the minimum feedback vertex set of the input graph.

## References


1 James A. Andrews and Michael S. Jacobson. On a generalization of chromatic number. *Congressus Numerantium*, 47:33–48, 1985.

2 Patrizio Angelini, Michael A. Bekos, Felice De Luca, Walter Didimo, Michael Kaufmann, Stephen G. Kobourov, Fabrizio Montecchiani, Chrysanthi N. Raftopoulou, Vincenzo Roselli, and Antonios Symvonis. Vertex-coloring with defects. *J. Graph Algorithms Appl.*, 21(3):313–340, 2017. doi:10.7155/jgaa.00418.

3 Dan Archdeacon. A note on defective colorings of graphs in surfaces. *J. Graph Theory*, 11(4):517–519, 1987. doi:10.1002/jgt.3190110408.

4 Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum $k$-plex problem. *Oper. Res.*, 59(1):133–142, 2011. doi:10.1287/opre.1100.0851.

5 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Parameterized (approximate) defective coloring. *SIAM J. Discret. Math.*, 34(2):1084–1106, 2020. doi:10.1137/18M1223666.

6 Rémy Belmonte, Michael Lampis, and Valia Mitsou. Defective coloring on classes of perfect graphs. *Discret. Math. Theor. Comput. Sci.*, 24, 2022. doi:10.46298/dmtcs.4926.

7 Nadja Betzler, Hans L. Bodlaender, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On making a distinguished vertex of minimum degree by vertex deletion. *Algorithmica*, 68(3):715–738, 2014. doi:10.1007/s00453-012-9695-6.

8 Nadja Betzler, Robert Bredereck, Rolf Niedermeier, and Johannes Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discret. Appl. Math.*, 160(1-2):53–60, 2012. doi:10.1016/j.dam.2011.08.013.

9 Nadja Betzler and Johannes Uhlmann. Parameterized complexity of candidate control in elections and related digraph problems. *Theor. Comput. Sci.*, 410(52):5425–5442, 2009. doi:10.1016/j.tcs.2009.05.029.

10 Marthe Bonamy, Lukasz Kowalik, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. Tight lower bounds for the complexity of multicoloring. *ACM Trans. Comput. Theory*, 11(3):13:1–13:19, 2019. doi:10.1145/3313906.

11 Glencora Borradaile and Hung Le. Optimal dynamic program for r-domination problems over tree decompositions. In *11th International Symposium on Parameterized and Exact Computation, IPEC 2016*, volume 63 of *LIPIcs*, pages 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.IPEC.2016.8.

12 Ilkyoo Choi and Louis Esperet. Improper coloring of graphs on surfaces. *J. Graph Theory*, 91(1):16–34, 2019. doi:10.1002/jgt.22418.

13 Lenore J. Cowen, Robert Cowen, and Douglas R. Woodall. Defective colorings of graphs in surfaces: Partitions into subgraphs of bounded valency. *J. Graph Theory*, 10(2):187–195, 1986. doi:10.1002/jgt.3190100207.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

15 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

16 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

17 Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010. doi:10.1016/j.tcs.2010.06.018.

18 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.

19 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. New algorithms for mixed dominating set. *Discret. Math. Theor. Comput. Sci.*, 23(1), 2021. doi:10.46298/dmtcs.6824.

20 Louis Dublois, Michael Lampis, and Vangelis Th. Paschos. Upper dominating set: Tight algorithms for pathwidth and sub-exponential approximation. *Theor. Comput. Sci.*, 923:271–291, 2022. doi:10.1016/j.tcs.2022.05.013.

21 Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of nemhauser and trotter's local optimization theorem. *J. Comput. Syst. Sci.*, 77(6):1141–1158, 2011. doi:10.1016/j.jcss.2010.12.001.

22 Jacob Focke, Dániel Marx, Fionn Mc Inerney, Daniel Neuen, Govind S. Sankar, Philipp Schepper, and Philip Wellnitz. Tight complexity bounds for counting generalized dominating sets in bounded-treewidth graphs. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 3664–3683. SIAM, 2023. doi:10.1137/1.9781611977554.ch140.

**23** Jacob Focke, Dániel Marx, and Pawel Rzazewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

**24** Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discret. Appl. Math.*, 86(2-3):213–231, 1998. doi:10.1016/S0166-218X(98)00035-3.

**25** Toshihiro Fujito. Approximating bounded degree deletion via matroid matching. In *Algorithms and Complexity – 10th International Conference, CIAC 2017*, volume 10236 of *Lecture Notes in Computer Science*, pages 234–246, 2017. doi:10.1007/978-3-319-57586-5_20.

**26** Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.66.

**27** Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/s00453-020-00758-8.

**28** Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight bounds for counting colorings and connected edge sets parameterized by cutwidth. In *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022*, volume 219 of *LIPIcs*, pages 36:1–36:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.36.

**29** Tesshu Hanaka, Ioannis Katsikarelis, Michael Lampis, Yota Otachi, and Florian Sikora. Parameterized orientable deletion. *Algorithmica*, 82(7):1909–1938, 2020. doi:10.1007/s00453-020-00679-6.

**30** Frédéric Havet, Ross J. Kang, and Jean-Sébastien Sereni. Improper coloring of unit disk graphs. *Networks*, 54(3):150–164, 2009. doi:10.1002/net.20318.

**31** Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. *Discret. Appl. Math.*, 327:33–46, 2023. doi:10.1016/j.dam.2022.11.011.

**32** Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structural parameters, tight bounds, and approximation for (k, r)-center. *Discret. Appl. Math.*, 264:90–117, 2019. doi:10.1016/j.dam.2018.11.002.

**33** Ioannis Katsikarelis, Michael Lampis, and Vangelis Th. Paschos. Structurally parameterized d-scattered set. *Discret. Appl. Math.*, 308:168–186, 2022. doi:10.1016/j.dam.2020.03.052.

**34** Michael Lampis. Parameterized approximation schemes using graph widths. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014*, volume 8572 of *Lecture Notes in Computer Science*, pages 775–786. Springer, 2014. doi:10.1007/978-3-662-43948-7_64.

**35** Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

**36** Bernt Lindström. On a combinatorial problem in number theory. *Canadian Mathematical Bulletin*, 8(4):477–490, 1965. doi:10.4153/CMB-1965-034-2.

**37** Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. *ACM Trans. Algorithms*, 14(2):13:1–13:30, 2018. doi:10.1145/3170442.

**38** Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Fpt-approximation for FPT problems. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 199–218. SIAM, 2021. doi:10.1137/1.9781611976465.14.

**39** Benjamin McClosky and Illya V. Hicks. Combinatorial algorithms for the maximum k-plex problem. *J. Comb. Optim.*, 23(1):29–49, 2012. doi:10.1007/s10878-010-9338-2.

40 Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *J. Comb. Optim.*, 24(3):347–373, 2012. doi:10.1007/s10878-011-9391-5.

41 Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of vertex cover. *Discret. Appl. Math.*, 152(1-3):229–245, 2005. doi:10.1016/j.dam.2005.02.029.

42 Karolina Okrasa and Pawel Rzazewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

43 Michael Okun and Amnon Barak. A new approach for approximating node deletion problems. *Inf. Process. Lett.*, 88(5):231–236, 2003. doi:10.1016/j.ipl.2003.08.005.

44 Venkatesh Raman, Saket Saurabh, and Sriganesh Srihari. Parameterized algorithms for generalized domination. In *Combinatorial Optimization and Applications, Second International Conference, COCOA 2008*, volume 5165 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2008. doi:10.1007/978-3-540-85097-7_11.

45 Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theor. Comput. Sci.*, 137(2):253–268, 1995.

46 Craig A. Tovey. A simplified np-complete satisfiability problem. *Discret. Appl. Math.*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

47 Bas A. M. van Geffen, Bart M. P. Jansen, Arnoud A. W. M. de Kroon, and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. *J. Graph Algorithms Appl.*, 24(3):461–482, 2020. doi:10.7155/jgaa.00542.

48 Johan M. M. van Rooij. A generic convolution algorithm for join operations on tree decompositions. In *Computer Science – Theory and Applications – 16th International Computer Science Symposium in Russia, CSR 2021*, volume 12730 of *Lecture Notes in Computer Science*, pages 435–459. Springer, 2021. doi:10.1007/978-3-030-79416-3_27.

49 Mingyu Xiao. A parameterized algorithm for bounded-degree vertex deletion. In *Computing and Combinatorics – 22nd International Conference, COCOON 2016*, volume 9797 of *Lecture Notes in Computer Science*, pages 79–91. Springer, 2016. doi:10.1007/978-3-319-42634-1_7.

50 Mingyu Xiao. On a generalization of nemhauser and trotter's local optimization theorem. *J. Comput. Syst. Sci.*, 84:97–106, 2017. doi:10.1016/j.jcss.2016.08.003.

# Massively Parallel Algorithms for the Stochastic Block Model

**Zelin Li**
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

**Pan Peng**
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

**Xianbin Zhu**
Department of Computer Science, City University of Hong Kong, Hong Kong, China

## Abstract

Learning the community structure of a large-scale graph is a fundamental problem in machine learning, computer science and statistics. Among others, the Stochastic Block Model (SBM) serves a canonical model for community detection and clustering, and the Massively Parallel Computation (MPC) model is a mathematical abstraction of real-world parallel computing systems, which provides a powerful computational framework for handling large-scale datasets. We study the problem of exactly recovering the communities in a graph generated from the SBM in the MPC model. Specifically, given $kn$ vertices that are partitioned into $k$ equal-sized clusters (i.e., each has size $n$), a graph on these $kn$ vertices is randomly generated such that each pair of vertices is connected with probability $p$ if they are in the same cluster and with probability $q$ if not, where $p > q > 0$.

We give MPC algorithms for the SBM in the (very general) *s-space MPC model*, where each machine is guaranteed to have memory $s = \Omega(\log n)$. Under the condition that[1] $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(k^{\frac{1}{2}} n^{-\frac{1}{2}+\frac{1}{2(r-1)}})$ for any integer $r \in [3, O(\log n)]$, our first algorithm exactly recovers all the $k$ clusters in $O(kr \log_s n)$ rounds using $\tilde{O}(m)$ total space, or in $O(r \log_s n)$ rounds using $\tilde{O}(km)$ total space. If $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(k^{\frac{3}{4}} n^{-\frac{1}{4}})$, our second algorithm achieves $O(\log_s n)$ rounds and $\tilde{O}(m)$ total space complexity. Both algorithms significantly improve upon a recent result of Cohen-Addad et al. [PODC'22], who gave algorithms that only work in the *sublinear space MPC model*, where each machine has local memory $s = O(n^\delta)$ for some constant $\delta > 0$, with a much stronger condition on $p, q, k$. Our algorithms are based on collecting the $r$-step neighborhood of each vertex and comparing the difference of some statistical information generated from the local neighborhoods for each pair of vertices. To implement the clustering algorithms in parallel, we present efficient approaches for implementing some basic graph operations in the $s$-space MPC model.

**2012 ACM Subject Classification** Theory of computation → Massively parallel algorithms

**Keywords and phrases** Massively Parallel Computation, Stochastic Block Model, Graph Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.78

**Related Version** *Full Version*: https://arxiv.org/abs/2307.00530

**Funding** *Zelin Li and Pan Peng*: Supported in part by NSFC grant 62272431 and "the Fundamental Research Funds for the Central Universities".
*Xianbin Zhu*: Supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 11213620].

**Acknowledgements** We would like to thank the anonymous reviewers for their detailed comments.

[1] $\tilde{\Omega}(\cdot)$ hides $\mathrm{poly}(\log kn)$ factors.

# 1 Introduction

Graph clustering is a fundamental task in machine learning, computer science and statistics. In this task, given a graph that may represent a social/information/biological network, the goal is to partition its vertex set into a few maximal subsets (called *clusters* or *communities*) of similar vertices. Depending on the context, a cluster may correspond to a social group of people with the same hobbies, a group of web-pages with similar contents or a set of proteins that interact very frequently. Intuitively, in a good clustering of a graph, there are few edges between different clusters while there are relatively many edges inside each cluster. There is no unified formalization on the notions of graph clustering and clusters. Here we focus on a natural and widely-used model for graph clustering, the *stochastic block model (SBM)*. In the SBM, we are given a set $V$ of $N = kn$ vertices such that there is a hidden partition of $V$ with $V = \bigcup_{i=1}^{k} V_i$, $V_i \cap V_j = \emptyset$ for any $1 \leq i < j \leq k$, where each set $V_i$ is called a *cluster* (or *community*). For simplicity, we assume that each cluster has an equal size, i.e., $|V_i| = n$. We say a graph $G = (V, E)$ is generated from the SBM with parameters $n, p, q, k$, abbreviated as $\mathrm{SBM}(n, p, q, k)$, if for any two vertices $u, v$ that belong to the same cluster, the edge $(u, v)$ appears in $G$ with probability $p$; for any two vertices $u, v$ that belong to two different clusters, the edge $(u, v)$ appears with probability $q$, where $0 < q < p < 1$.

Thanks to its simplicity and its ability in explaining the community structures in real world data, the SBM has been extensively studied in the computer science literature. Most previous work has been focusing on algorithms that work on a single machine, with the goal of extracting the communities with the optimal (computational and/or statistical) trade-offs between parameters $n, p, q, k$, for different types of recoveries (i.e., exact, weak, and partial recovery). Significant progress has been made on such algorithms (and their limitations) in the past decades (see the survey [1]). However, most of these algorithms are essentially sequential and cannot be adapted to the parallel or distributed environment, which is unsatisfactory as modern graphs are becoming massive and most of them cannot be fitted into the main memory of a single machine.

We study the problem of exactly recovering communities of a graph from the SBM in the massively parallel computation (MPC) model [24, 22, 5], which is a mathematical abstraction of modern frameworks of real-world parallel computing systems like MapReduce [19], Hadoop [28], Spark [29] and Dryad [23]. In this model, there are $M$ machines that communicate in synchronous rounds, where the local memory of each machine is limited to $s$ words, each of $O(\log n)$ bits. A word is enough to store a node or a machine identifier from a polynomial (in $n$) domain. Communication is the largest bottleneck in the MPC model. Take the graph problem as an example. The edges of the input graph are arbitrarily distributed across the $M$ machines initially. Ideally, we would like to use minimal number of rounds of computation while using small (say sublinear) space per machine and small total space (i.e., the sum of space used by all machines).

Recently, Cohen-Addad et al. [16] gave two algorithms Majority and Louvain that recover the communities in a graph generated from $\mathrm{SBM}(n, p, q, k)$ when $\frac{p-q}{\sqrt{p}} \geq \Omega(n^{-\frac{1}{4}+\varepsilon})$ and $k$ is constant. They work in $O(\frac{1}{\varepsilon \cdot \delta})$ rounds in the *sublinear space* MPC model, i.e., each machine has local memory $s = O(n^{\delta})$, for any constant $\delta > 0$. Their algorithms and analysis improve upon previous sequential versions of Majority and Louvain given by Cohen-Addad et al [12]. Note that for any sequential algorithm, it is known that $\frac{p-q}{\sqrt{p}} = \Omega(\sqrt{\frac{\log n}{n}})$ is necessary for exact recovery even for $k = 2$ [2]; there exist spectral algorithms and SDP-based algorithms that find all clusters and achieve this parameter threshold [1]. Therefore, it is natural to ask *if one can obtain a round-efficient MPC algorithm in the sublinear space model with roughly the same parameter threshold.*

In this paper, we consider a more general setting that we call *the s-space MPC model* in which the local memory $s$ is only guaranteed to satisfy that $s = \Omega(\log n)$. Nowadays, the growth rate of data volume far exceeds the growth rate of machine hardware storage and it is likely that we need much more machines to analyze large-scale data. Furthermore, the problem of clustering of data points from some metric space on such a model has recently received increasing interest [8, 20, 4, 14, 17, 18] (see also Section 1.3), partly due to the fact that in some scenarios, the number of clusters $k$ is too large such that even just storing $k$ representatives of all the clusters is not possible in a single machine. Note that this model is more difficult to handle than the sublinear space model, and we need to carefully partition the data across machines so that different machines work in different "regions of space" to get a good tradeoff between communication and the used space. Here, we are interested in the question *whether we can obtain an SBM clustering algorithm in the s-space MPC model with good tradeoffs between communication, space and SBM parameters.*

## 1.1 Our Results

We give clustering algorithms for the SBM that work in the $s$-space MPC model where the local memory $s$ of each machine is only guaranteed to satisfy that $s = \Omega(\log n)$. Let $m = \Theta(kn^2p + k^2n^2q)$ denote the total number of edges of the graph (our conditions always imply that $p \geq \Omega(\frac{\log n}{n})$ and $k \leq n$). We use "with high probability" to denote "with probability at least $1 - O(n^{-1})$".

Our first algorithm has the following performance guarantee.

▶ **Theorem 1.** *Let $r$ be any integer such that $3 \leq r \leq O(\log n)$. Let $p, q \leq 0.75$ be parameters such that $\max\{p(1-p), q(1-q)\} \geq C_0 \log n/n$ where $C_0 > 0$ is some constant. Suppose that $\frac{p-q}{\sqrt{p}} \geq \Omega\left(k^{\frac{1}{2}} n^{-\frac{1}{2}+\frac{1}{2(r-1)}} \log^7(kn)\right)$. Let $G$ be a random graph generated from SBM$(n, p, q, k)$. Then there exists an algorithm in the s-space MPC model that outputs $k$ clusters in $O(kr \log_s n)$ rounds with high probability where each machine has $s = \Omega(\log n)$ memory. The total space used by the algorithm is $\tilde{O}(m)$.*

We note that the round complexity can be improved to be $O(r \log_s n)$ at the cost of increasing the total space by a $k$ factor, which is formalized in the following theorem.

▶ **Theorem 2.** *Under the same condition in Theorem 1, there exists an algorithm that outputs $k$ clusters in $O(r \log_s n)$ rounds where each machine has $s = \Omega(\log n)$ memory with high probability and uses $\tilde{O}(km)$ total space.*

Note that for any integer constant $3 \leq r \leq o(\log n)$ and any $k \leq \text{poly}(\log n)$, the round complexity of the above algorithm is $O(\log_s n)$ while the total space is $\tilde{O}(m)$. When $r = \Theta(\log n)$, then the recovery condition becomes $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(\sqrt{\frac{k}{n}})$, which almost matches the statistical limit in the sequential setting up to logarithmic terms [2]. In this case, our algorithm has round complexity $O(\log n \log_s n)$ for any $k \leq \text{poly}(\log n)$ in the $s$-space MPC model.

When the gap between $p, q$ is sufficiently large, we can achieve $O(\log_s n)$ rounds using $\tilde{O}(m)$ total space, i.e., both the round complexity and the total space complexity are independent of the number $k$ of clusters. Formally, we have the following theorem.

▶ **Theorem 3.** *Given a random graph $G$ from SBM$(n, p, q, k)$ with $\frac{p-q}{\sqrt{p}} \geq \Omega\left(k^{\frac{3}{4}} n^{-\frac{1}{4}} (\log n)^{\frac{1}{4}}\right)$, there exists an algorithm in the s-space MPC model that can output $k$ hidden clusters within $O(\log_s n)$ rounds with high probability, where $s = \Omega(\log n)$, and uses $\tilde{O}(m)$ total space.*

We note that all the algorithms in Theorem 1, 2 and 3 significantly improve the results of [16], of which the algorithms only work in the sublinear space MPC model, i.e., $s = O(n^{\delta})$ for some constant $\delta > 0$, and finish in $O(\frac{1}{\delta\varepsilon})$ rounds, assuming that $\frac{p-q}{\sqrt{p}} \geq n^{-1/4+\varepsilon}$ and $k$ is a constant. In both sublinear space and $s$-space models, our algorithms work for a much wider class of SBM graphs (i.e., the requirement on the conditions of $p, q, k$ are much weaker) than those in [16]. Furthermore, even for the same regime of parameters, our algorithms have better round complexity. For example, in the sublinear space MPC model, our round complexity (from Theorem 3) is $O(1/\delta)$ under the condition that $\frac{p-q}{\sqrt{p}} \geq \Omega(n^{-\frac{1}{4}}(\log n)^{\frac{1}{4}})$ and $k$ is constant, while the algorithms in [16] have round complexity $O(\frac{\log n}{\delta \log\log n})$ under the same condition[2].

Our algorithms are quite different from those in [16], in which the algorithms are based on the local-search methods and proceed in rounds by updating the so-called *swap values* for each node to decide where to move the node. Our algorithms are based on collecting the $r$-step neighborhood of each vertex and comparing the difference of some statistical information generated from the local neighborhoods for each pair of vertices.

To implement the above MPC algorithms, we give new algorithms of some basic graphs operations in the $s$-space MPC model in Section 3, including **RandomSet** (for randomly sampling a set), **ReorganizeNBR** that is for organizing the neighborhood of any two nodes $u, v$ in a set so that they are "aligned", i.e., the $i$-th byte of $u$ (or $v$) indicates whether the $i$-th node is the neighbor of $u$ (or $v$ ). We believe these results will be useful as basic tools in designing algorithms for other problems in the $s$-space MPC model.

## 1.2 Our Techniques

Our MPC algorithms are based on two simple sequential algorithms. We first describe our first algorithm given in Theorem 3. It is based on the observation that if $\frac{p-q}{\sqrt{p}} \geq \tilde{\Omega}(k^{\frac{3}{4}} n^{-\frac{1}{4}})$, then the number of common neighbors of any two vertices can be used to distinguish if they belong to the same cluster or not. That is, if $u, v$ belong to the same cluster, then the number of their common neighbors is above some threshold $\Delta$; otherwise, the number of common neighbors is smaller than $\Delta$. Let $N(v)$ denote the set of all the neighbors of $v$. We further note that to get $k$ clusters of $V$, it is not necessary to compute $|N(u) \cap N(v)|$ for *all pairs* of $u, v$ in $V$, which may cause too much communication for MPC implementation. Instead, we first randomly sample a small set $S'$ with $|S'| = \Theta(k \log n)$. Then we find $k$ representatives of the hidden clusters from $S'$ by computing $|N(u) \cap N(v)|$ for all pairs of $u, v$ in $S'$ and update $S'$ to be the set of $k$ representatives. Then we sample independently another small set $S$ of vertices, and find $k$ sub-clusters from $S$ by computing $|N(u) \cap N(v)|$ for $u \in S$ and $v \in S'$. (A set $T \subseteq S$ is called a *sub-cluster* of some cluster $V_i$ if $T \subseteq V_i$.) Based on the $k$ sub-clusters obtained from $S$, we can find all the hidden clusters $V_1, \ldots, V_k$ putting any vertex $v \in V \setminus S$ to the sub-cluster that contains the most number of neighbors of $v$.

There are several challenges to implementing the above algorithm in the $s$-space MPC model in which the local memory only satisfies that $s = \Omega(\log n)$. Note that in this model, even just to compute the number of common neighbors $|N(u) \cap N(v)|$ for any *fixed pair* $u, v$ in a few parallel rounds (say $O(\log_s n)$ rounds) is non-trivial. The reason is that the neighborhoods $N(u), N(v)$ can be much larger than $s$ and some neighborhoods will be used too many times which leads to large round complexity. To efficiently compute $|N(u) \cap N(v)|$ for $u \in S$ and $v \in S'$, we first show how to reorganize $N(u)$ and $N(v)$ for $u \in S$ and $v \in S'$

[2] This can be seen by setting $\frac{1}{\varepsilon} = \Theta(\frac{\log n}{\log\log n})$ in [16].

so that each byte of $N(u)$ and $N(v)$ for any two nodes aligned; then we can show how to compute $|N(u) \cap N(v)|$ in parallel efficiently by appropriately making some copies of $N(u)$ and $N(v)$. For these tasks, we give detailed MPC implementations of some basic operations, e.g., a procedure for copying neighbors of some carefully chosen nodes and aligning their neighbors while using no more than $\tilde{O}(m)$ total space.

Our MPC algorithms from Theorem 1 and 2 are based on a recent sequential algorithm given in [25]. Roughly speaking, one can use the power iterations of some matrix $B = A - q \cdot J$ to find the corresponding clusters, where $A$ is the adjacency matrix of the graph and $J$ is the all-1 matrix. It is shown that with high probability, the $\ell_2$-norm of $B_u^r - B_v^r$ is relatively small, if $u, v$ belong to the same cluster; and is large, otherwise. Here $B_u^r$ is the row corresponding to vertex $u$ in the matrix $B^r$. We show that in order to compute $\|B_u^r - B_v^r\|_2$, it suffices to compute the expressions $\mathbf{1}_x^T (A - qJ)^{2r} \mathbf{1}_y$ for all $x, y \in \{u, v\}$. To do so, we expand the above expression so that we get a sum of terms, each being a vector-matrix-vector multiplication. Then we give a combinatorial explanation of each term, and then calculate it in parallel efficiently based on some basic graph operations in the $s$-space model.

## 1.3 Related Work

There is a line of research on metric clustering in the MPC model. In this setting, the input is a set of data points from some metric space (e.g., Euclidean space), and the goal is to find $k$ representative centers, such that some objective function (e.g. the cost functions of $k$-means, $k$-median and $k$-center) is minimized (e.g., [8]). Bhaskara and Wijewardena [8] developed an algorithm that outputs $O(k \log k \log n)$ centers whose cost is within a factor of $O((\log n \log\log n)^2)$ of the optimal $k$-means (or $k$-median) clustering, using a memory of $s \in \Omega(d \log n)$ per machine and $O(\log_s n)$ parallel rounds. Note that this does not require $\Omega(k)$ memory per machine. Coy et al. [18] recently improved the approximation ratio of the algorithm for $k$-center in [8] to $O(\log^* n)$. Cohen-Addad et al. [17] gave a fully scalable $(1+\varepsilon)$-approximate $k$-means clustering algorithm when the instance exhibits a "ground-truth" clustering structure, captured by a notion of "$O(\alpha)$-perturbation resilient", and it uses $O(1)$ rounds and $O_{\varepsilon,d}(n^{1+1/\alpha^2+o(1)})$ total space with arbitrary memory per machine, where each data point is from $\mathbb{R}^d$.

Regarding the power method for SBM, Wang et al. [27] proposed an iterative algorithm that first employs the power method with a random starting point and then turns to a generalized power method that can find the communities in a finite number of iterations. Their algorithm runs in nearly linear time and can exactly recover the underlying communities at the information-theoretic limit. Cohen-Addad et al. [15] further gave a linear-time algorithm that recovers exactly the communities at the asymptotic information-theoretic threshold. Their algorithm is based on similar ideas as in [16], that is, given a partition, moving a vertex from one part to the part where it has most neighbors should somewhat improve the quality of the partition.

Correlation clustering has been studied under the MPC model. In this problem, a signed graph $G = (V, E, \sigma)$ is given as input, and the goal is to partition the vertex set into arbitrarily many clusters so that the disagreement of the corresponding clustering is minimized, where the disagreement is the number of edges that cross different clusters plus the number of non-adjacent pairs inside the clusters [9, 11, 26, 21, 10, 13, 3]. The state-of-the-art is a $(3+\varepsilon)$-approximation algorithm in $O(1/\varepsilon)$ rounds in the massively parallel computation (MPC) with sublinear space [7].

## 2 Preliminaries

Consider an undirected graph $G = (V, E)$ where $V$ is the set of vertices, and $E$ is the set of edges. We use $n$ to denote the size of $|V|$ and $m$ to denote the size of $|E|$. Each node in $G$ has a unique ID from 1 to $n$. We use $\mathsf{ID}(u)$ to denote the ID for a node $u \in V$. We use $d(u)$ to denote the degree of $u \in V$. Let $N(u)$ denote the set of neighbors of a node $u \in V$. Given a vertex set $S \subset V$, we use $G[S]$ to denote the subgraph induced by vertices in $S$. In this paper, we abuse the use of node(s) and vertex(vertices). We use $[i]$ to denote $\{1, 2, \cdots i\}$. When nodes are active (inactive), they execute (do not execute) algorithms.

### Chernoff Bound

Let $X_1, ..., X_n$ be independent binary random variables, and $X = \Sigma_{i=1}^{n} X_i$, and $\mu = \mathbb{E}[X]$. Then it holds that for all $\delta > 0$ that $\mathbb{P}[X \geq (1+\delta)\mu] \leq (\frac{e^{-\delta}}{(1+\delta)^{1+\delta}})^{\mu} \leq e^{-\min[\delta^2, \delta]\mu/3}$; For all $\delta \in (0, 1)$, $\mathbb{P}[X \leq (1-\delta)\mu] \leq (\frac{e^{\delta}}{(1-\delta)^{1-\delta}})^{\mu} \leq e^{-\delta^2\mu/2}$.

### The MPC model

In this model, we assume that all data is arbitrarily distributed among some machines. Let $N$ denote the total amount of data. Each machine has local memory $s$. In our settings, $s = \Omega(\log n)$. The sum of all local memory is $N = O((m+n)\text{poly}(\log n))$. The communication between any pair of two machines is synchronous, and the bandwidth is $s$ words.

We ignore the cost of local communication and computation happening in each machine. As the description of the MPC model in the literature, we ignore some communication details among different machines and suppose that all machines are known to each other which means that any machine can send messages to another machine directly (even when the local memory is very small). For the problems in the MPC model, we aim to make the total number of communication rounds among machines as small as possible.

▶ **Definition 4** (separable function). *Let $f : 2^{\mathbb{R}} \to \mathbb{R}$ denote a set function. We say that $f$ is separable if and only if for any set of reals $A$ and for any $B \subseteq A$, we have $f(A) = f(f(B), f(A \setminus B))$* [3].

▶ **Lemma 5** ([6]). *Given an $n$-vertex graph, we have $x_u$ for each node $u \in V$. If the function $f$ is a separable function, then there exists an algorithm that computes $f(\{x_i \in N(u)\})$ for each $u \in V$ with high probability in the sublinear space MPC model in $O(1/\delta)$ rounds using $\tilde{O}(m)$ space where each machine has space $O(n^{\delta})$.*

In the $s$-space MPC model, we restate the following folklore lemma.

▶ **Lemma 6.** *Given an $n$-vertex graph, there exists an algorithm that makes each node $u \in V$ visit $N(u)$ in $O(\log_s n)$ rounds with high probability.*

The sorting algorithm is a very important black-box tool in the MPC model, which is stated as follows.

▶ **Theorem 7** ([22]). *Sorting can be solved in $O(\log_s n)$ rounds in the $s$-space MPC model.*

[3] For example, $f$ can be a sum function.

Furthermore, it has been shown that indexing and prefix-sum operation can be performed in $O(\log_s n)$ rounds [22]. We refer to the INDEX Algorithm for solving indexing problems in the $s$-space MPC model and the SORTING Algorithm for solving sorting problems in the same model. Throughout the context, we will rely on the fundamental properties associated with the aforementioned operations, as well as Lemma 5 and Lemma 6 by default.

## 3 Implementing Basic Graph Operations in the $s$-space MPC Model

In this section, we present algorithms for several fundamental graph operations in the $s$-space model, which will be utilized in our MPC algorithms. To the best of our knowledge, most of these operations have not been previously implemented in the $s$-space MPC model. We denote the machines holding node $x$ as $M_x$. (It is important to note that the MPC model follows an edge-partition model, which means that multiple machines may hold the same vertices). It is worth mentioning that in order to implement some of our proposed algorithms, we utilize previous algorithms for basic MPC operations, as demonstrated in the full version.

### RandomSet

In the RandomSet problem, given an input value $X = \Omega(\log n)$, our goal is to output a random set $S = \{x_1, x_2, \ldots, x_{|S|}\}$ where each element $x_i$ ($i \in [|S|]$) is selected uniformly and independently at random and $S' = \{(x, y)|x \in S \cap M_x, y = \mathrm{Ind}_S(x)\}$ where $|S| = \Theta(X)$, and $\mathrm{Ind}_S(x)$ is the index of of $x \in S$ in $S$. We use the algorithm RANDOMSET to solve the RandomSet problem.

▶ **Lemma 8.** *The RandomSet problem can be solved in the $s$-space MPC model in $O(\log_s n)$ rounds where $s = \Omega(\log n)$.*

### ReorganizeNBR

The ReorganizeNBR problem involves an input set $S$, where the objective is to reorganize $N(u)$ for all $u \in S$ in a manner that *aligns* the bytes of $N(u)$ and $N(v)$ for any two nodes $u, v \in S$. Specifically, the $i$-th byte of $N(u)$ indicates whether the $i$-th node is a neighbor of $u$. The motivation behind the ReorganizeNBR problem is to efficiently compute $|N(u) \cap N(v)|$ for any pair of nodes $u$ and $v$ in $S$ (refer to Figure 1 for illustration). We utilize the REORGANIZENBR algorithm to address this problem.

Now, we show how to reorganize a graph in MPC model where each machine has memory $s = \Omega(\log n)$. We say a subgraph $H$ is a randomly sampled subgraph if nodes in $V_H$ are randomly sampled, and $H$ is the set of edges and vertices constructed from picking $V_H$ with their incident edges. We use $H = (V_H, E_H)$ to denote such a random sampled graph.

▶ **Lemma 9.** *Given a graph $G = (V, E)$ with $m = \Theta(n^{c+1})$, where $m$ is the number of edges and $n$ is the number of vertices, and $c \in (0, 1]$ is some positive constant, for a given randomly sampled subgraph $H = (V_H, E_H)$ satisfying $|V_H| \leq \tilde{O}(m/n)$ where $V_H$ is constructed by RANDOMSET, there exists an algorithm that can reorganize $N(u)$ for each $u \in V_H$ in constant rounds, with total space complexity $\tilde{O}(m)$ in the MPC model where each machine has a memory of $s = \Omega(\log n)$.*

### CopyNBR($\mathcal{S}, t$)

Suppose we have $\mathcal{S} = \{N(v_1), \ldots, N(v_x)\}$ stored in machines where $v_i \in S$, $i \in [x]$ and $S$ is a random set created by RANDOMSET. We will create $N(v_i)_1, \ldots, N(v_i)_t$ for each $N(v_i)$ where $i \in [x]$. The goal is to make $t$ copies of $N(u)$ for each $u \in S$ such that we can execute

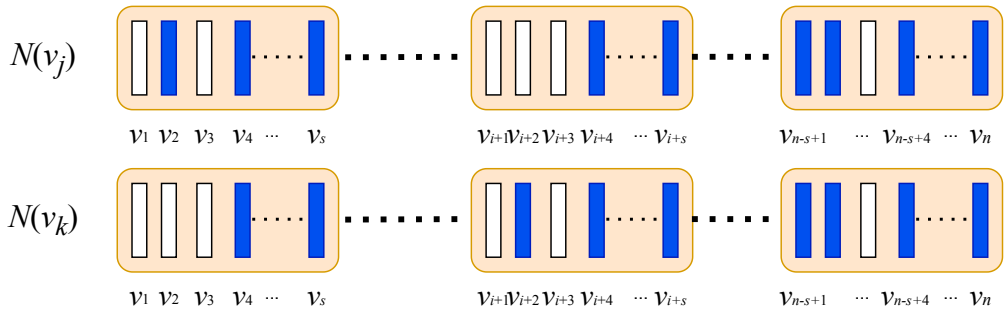



**Figure 1** Align Operation. The first row represents $N(v_j)$, the blue rectangle (treated as "1") indicates that the corresponding vertex $v_i \in N(v_j)$, and the white rectangle (treated as "0") indicates that the vertex $v_i \notin N(v_j)$. Once $N(v_j)$ and $N(v_k)$ are encoded as such bit strings, we can compare the strings simultaneously to compute their common neighbors.

other algorithms in parallel. In our setup, each $N(u)$ where $u \in S$ is organized in a collection of consecutive machines. To solve this problem, we employ the CopyNBR$(\mathcal{S}, t)$ algorithm. Upon executing CopyNBR$(\mathcal{S}, t)$, all $t$ copies of $\mathcal{S}$ are stored in consecutive machines.

▶ **Lemma 10.** *The CopyNBR$(\mathcal{S}, t)$ problem can be solved in $O(\log_s n)$ rounds in the $s$-space MPC model where $s = \Omega(\log n)$ and $t$ is a parameter satisying $n|S|t \leq \tilde{O}(m)$.*

### EvenCluster

Consider a set $S$ comprising nodes labeled from 1 to $k$. The objective is to ensure that the number of nodes with labels in $S$ is even. To achieve this, we employ the EvenCluster algorithm, designed specifically to solve this problem.

▶ **Lemma 11.** *In the MPC model with each machine's memory $s = \Omega(\log n)$, there exists an algorithm that can output $S' \subseteq S$ within $O(\log_s n)$ rounds, such that each label in $S'$ is associated with the same number of nodes with that label.*

### RepresentativeK($S$)

In the RepresentativeK$(S)$ problem, the input is a set $S = S_1 \cup \cdots \cup S_k$ of nodes with $|S|$ labels (each node in $S_i$ has $|S_i|$ labels) where $S_i \cap S_j = \emptyset$ for any $i \neq j$ and $|S_i| = \Theta(|S|/k) \geq \Omega(\log n)$. Our goal is to output $k$ nodes with $k$ representative labels. We use RepresentativeK$(S)$ to solve this problem.

▶ **Lemma 12.** *The RepresentativeK$(S)$ problem can be solved in $O(\log_s n)$ rounds where $S$ is created by RandomSet and $|S| \geq \Omega(k \log n)$ in the $s$-space MPC model ($s = \Omega(\log n)$).*

### CompareCut($S, V$)

In the CompareCut$(S, V)$ problem, the input is a set $S$ of $k$ sets, i.e., $\{S_1, S_2, \ldots, S_k\}$ and the vertex set $V$, the goal is to output the largest one among numbers $n(u, S_i)$ of edges between $u \in V$ and $S_i$ for each $S_i$, along with the label of $u$ (i.e., the label of the $S_i$), where $i \in [k]$. We use ComputeCut(S,V) to solve this problem.

▶ **Lemma 13.** *Given a graph $G = (V, E)$, let $S \subset V$ be a random set of nodes created by RandomSet. The CompareCut$(S, V)$ problem can be solved in $O(\log_s n)$ rounds in the $s$-space MPC model ($s = \Omega(\log n)$).*

# 4 The Algorithm Based on Neighbor Counting

Recall that a graph $G = (V, E)$ is generated from the SBM$(n, p, q, k)$ if there is a hidden partition $V = \cup_{i=1}^{k} V_k$ of the $nk$-vertex set $V$, and for any two vertices $u, v$ that belong to the same cluster, the edge $(u, v)$ appears in $E$ with probability $p$; for any two vertices $u, v$ that belong to two different clusters, the edge $(u, v)$ appears in $E$ with probability $q$, where $0 < q < p < 1$. In this section, we give the algorithm underlying Theorem 3.

We first give a simple sequential algorithm based on comparing common neighbors. Then we show how to implement it in the $s$-space model. To do so, we give implementations of a number of basic graph operations in the $s$-space model, which is deferred to Section 3.

## 4.1 A Sequential Algorithm Based on Counting Common Neighbors

▶ **Theorem 14.** *If* $\frac{p-q}{\sqrt{p}} \geq \Omega(\frac{(k+1)^{1/2}}{n^{1/4}})$*, the algorithm* CommNBR *can output* $k$ *clusters in* $O(\frac{k^2 n \log n}{p})$ *time with probability* $1 - O(\frac{1}{n})$.

In our sequential algorithm CommNBR, we first randomly sample a set $S$ of $\frac{21nk^2 \log n}{d}$ nodes from $V$ such that each cluster has more than $\Theta(\log n)$ nodes with high probability where $d$ is the number of neighbors of an arbitrary node $u \in V$. Then, for each pair $u, v$, we count the number of their common neighbors in $G$, i.e., those vertices that are connected to both $u$ and $v$. If the number of common neighbors is above some threshold $\Delta$, then we put them into the same cluster. In this way, we can obtain $k$ *sub-clusters* of $S$, $C_1, \ldots, C_k$. That is, each $C_i \subseteq S$ and is a subset of some cluster, i.e., $C_i = V_{\pi(i)} \cap S$ for some permutation $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$. Let $L(v, C_i)$ denote the number of incident edges between a node $v$ and a cluster $C_i$. We can then cluster each remaining node $v \in V \setminus S$ by finding the index $j$ such that $L(v, C_j)$ is the greatest among all numbers $L(v, C_i), 1 \leq i \leq k$.

For the intuition of the existence of such a threshold $\Delta$, let us take the case $k = 2$ as an example. In this case, for any two vertices $u, v$ belonging to the same cluster, the expected number of common neighbors is $p^2 n + q^2 n$; for any two vertices $u, v$ belonging to two different clusters, the expected number of common neighbors is $2npq$. Since $\frac{p-q}{\sqrt{p}} \geq \Omega\left(\frac{(k+1)^{1/2}}{n^{1/4}}\right)$, there exists a sufficiently large gap between these two numbers so that we can define a suitable threshold. However, the values of $p$ and $q$ are not provided. To address this issue, we propose an algorithm called ComputeDEL, which can be described as follows. We first sample a set $\mathcal{S}_\Delta$ of $\Theta(k \log n)$ nodes, and for each pair $u, v \in \mathcal{S}_\Delta$, we compute a set $\mathcal{V}_\Delta$ of values of $|N(u) \cap N(v)|$. We let $\Delta' = \max\{value \in \mathcal{V}_\Delta\}$, and then we have $\Delta = \Delta' - 9\sqrt{\Delta' \log n}$.

## 4.2 Implementation in the $s$-space MPC model

Now we describe our MPC algorithm MPC-CommNBR, which is an implementation of CommNBR, where the local memory is $s = \Omega(\log n)$, and prove Theorem 3.

Recall that in CommNBR, there are two major steps. In the first step, we need to find $k$ clusters from a set $S$ of randomly sampled nodes. In the second step, based on the clustering on $S$, we cluster all nodes in $V$. The major challenge lies in the simulation (in MPC model) of the first step which is to compare common neighbors between two nodes $u$ and $v$. It is easy to see that computing $|N(u) \cap N(v)|$ for $u, v \in V$ is exactly the task of finding common elements in two sets. For convenience, we use a set $\mathsf{S}_u$ to denote $N(u)$ for a node $u \in V$. Then, we need to find the common elements between $\mathsf{S}_u$ and $\mathsf{S}_v$ by a method $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$. Note that we need to execute $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$ for different $u$ and $v$ for many times. Therefore, to compute $|N(u) \cap N(v)|$ for different $u, v \in V$ efficiently, we need to solve two problems.

The first one is to implement $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$ efficiently in MPC model where each machine's memory is $s = \Omega(\log n)$. The second one is to execute the first one in parallel. For the first one, we use a simple method to implement $\textsc{comm}(\mathsf{S}_u, \mathsf{S}_v)$ in MPC model. Let $V'$ be the set of nodes in which for each $u \in V'$, $\mathsf{S}_u$ will be compared. We make each byte of $\mathsf{S}_u (u \in V')$ aligned. Then, we can directly compute $|\mathsf{S}_u \cap \mathsf{S}_v|$. For the second one, we solve it by copying sets for enough times and then we let machines storing these copied sets execute the same algorithm in parallel. We use the MPC implementations of the basic graph operations in Section 3 to implement our clustering algorithm here.

**Algorithm 1** MPC-CommNBR: An MPC algorithm based on counting common neighbors.

**Input:** A SBM graph $G$;

1: Let $d = |N(u)|$ where $u$ is an arbitrary node in $V$
2: Apply RandomSet to obtain random node set $S'$ and $S$, where $|S'| = \Theta(k \log n)$ and $|S| = (21k^2 n \log n)/d$
3: Update $S'$ and obtain $\Delta$ by ComputeREP($S'$)
4: Obtain $k$ sub-clusters $S_1, \ldots, S_k$ by ComputeSubcluster($S, S', \Delta$)
5: Obtain $k$ clusters of $V$ by ComputeCluster($S_1, \ldots, S_k, V$).

**Algorithm 2** ComputeRep: Compute representative for each cluster by common neighbors and $\Delta$.

**Input:** Random vertex set $S'$;

1: Run CompareINIT($S'$);
2: Obtain $k$ nodes with $k$ representative lables by RepresentativeK($S'$);
3: Update $S'$ by only keeping $k$ nodes obtained from Step 2.
4: **return** $S'$ and $\Delta$

For the algorithm CompareINIT($S'$), we describe it as follows.

CompareINIT($S'$):
**1)** Reorganize neighbors of nodes in $S'$ by ReorganizeNBR($S'$).
**2)** Execute CopyNBR($S'$) to create $|S'|$ copies of $N(u)$ for each $u \in S'$.
**3)** Based on copies of $N(u)$ from 2), we directly compute $|N(u) \cap N(v)|$ in parallel where $u, v \in S'$.
**4)** Execute CompareGRP to obtain final results by summing all partial results obtained from 3).
**5)** Compute $\Delta'$, i.e., the maximum value of $|N(u) \cap N(v)|$ for all pairs of $u, v \in S'$ and output $\Delta = \Delta' - 9\sqrt{\Delta' \log n}$.

In ComputeSubcluster($S', S, \Delta$), the process is similar to ComputeREP($S'$). The major difference is that we only copy $N(u)$ for each $u \in S$ for $k$ times and copy $N(v)$ for each $v \in S'$ for $|S|$ times. By computing $|N(u) \cap N(v)|$ where $u \in$ copy-$S$, $v \in$ copy-$S'$, and copy-$S$, copy-$S'$ are the copies of $S$ and $S'$, we can obtain $k$ sub-clusters of $S$. The details of computing can refer to ComputeREP($S'$).

In ComputeCluster($S_1, \ldots, S_k, V$), we first apply EvenCluster($S$) to output $k$ sub-clusters from $S$ such that each cluster has the same number of nodes. Then, we use ComputeCut($S, V$) to cluster $V$.

Next, we show the details of CompareGRP used in the procedure of ComputeREP($S'$). In the CompareGRP problem, the input is a set of groups of machines and the goal is to output the results by comparing groups of machines. Take two groups $A$, $B$ of machines as

an example. Our goal is to output the common elements by comparing $A$ and $B$. We say that group $A$ compares to group $B$ which means that the $i$-th member machine of the group $A$ will compare to the $i$-th member machine of group $B$ ($i \in [n/s]$).

▶ **Lemma 15.** *Given a set of consecutive groups each of which has $n/s$ member machines, there exists an algorithm that takes $O(\log_s n)$ rounds to obtain the results of comparing data between groups correspondingly.*

Now we are ready to prove Theorem 3.

**Proof of Theorem 3.** The correctness of obtaining $k$ clusters based on counting common neighbors can be seen in Theorem 14. By Lemma 8, we can create randomly sampled sets $S$ and $S'$ such that each machine $M$ knows indexes of nodes in $M \cap S$ and $M \cap S'$ within $O(\log_s n)$ rounds.

Next, we first prove that by ComputeREP($S'$), we can obtain $k$ sub-clusters within $O(\log_s n)$ rounds. By Lemma 9, it takes $O(\log_s n)$ rounds for ReorganizeNBR($S'$). By Lemma 10, we use $O(\log_s n)$ rounds to finish CopyNBR($S'$) for each $N(u)$ where $u \in S'$. By Lemma 15, it takes $O(\log_s n)$ to first get partial results and then obtain the complete results of $|N(u) \cap N(v)|$ where $u, v \in S'$. We can use $O(\log_s n)$ rounds to obtain $\Delta$ by simulating ComputeDEL within $O(\log_s n)$ rounds. Then by Lemma 15 and Lemma 12, we can obtain $k$ sub-clusters from $S'$ in $O(\log_s n)$ rounds in the MPC model and the total space is $\tilde{O}(m)$. Similarly, by ComputeSubcluster($S', S, \Delta$), we can prove that within $O(\log_s n)$ rounds, we can obtain $k$ clusters from $S$ in the MPC model and the total space used is $\tilde{O}(m)$.

Now, let us see the last step of obtaining $k$ clusters of $V$. By Lemma 11 and setting $|S^*| = \frac{20k^2 n \log n}{d}$, we can output $S^* \subseteq S$ such that for any two labels $i, j \in [k]$, we have $\mathsf{N}_{S^*}(i) = \mathsf{N}_{S^*}(j)$ in $O(\log_s n)$ rounds, where $\mathsf{N}_{S^*}(i)$ is the number of nodes in $S^*$ with label $i$. Finally, by Lemma 13, we can decide all labels of $V$ within $O(\log_s n)$ rounds with high probability. The total space used in MPC model is $\tilde{O}(m)$. Thus, our proof is completed. ◀

## 5 The Algorithm Based on Power Iterations

In this section, we give another MPC algorithm for a general SBM graph in the $s$-space model and prove Theorem 1. The omitted proof of this section is deferred to the full version. Also, we first carefully design a sequential algorithm and then we implement it in the $s$-space MPC model. Our second sequential algorithm is based on power iterations which perform well in a recent algorithm in [25]. The algorithm makes use of the adjacency matrix $A$ of the graph $G$, from which we define a matrix $B = A - q \cdot J$, where $J$ is the $n \times n$ all-1 matrix. Then it decides if two vertices $u, v$ are in the same cluster or not by checking the $\ell_2$-norm of the difference between $B_u^r$ and $B_v^r$, which are the rows corresponding to vertices $u, v$, respectively, in the matrix $B^r$ (the $r$-th power of matrix $B$).

We note that the algorithm in [25] only considers the special case that $r = \log n$. Here, our sequential algorithm considers all possible $r \in \{1, \ldots, O(\log n)\}$ and for each $r$ we choose a different threshold $\Delta$, which depends on the value of $p, q$ and $k$. To implement our sequential algorithm in the MPC model, we divide the process of matrix computation into different components each of which can be implemented efficiently in the $s$-space model.

### 5.1 A Sequential Algorithm Based on Power Iterations

We now describe Algorithm PowerIteration. Let $A$ be an adjacency matrix of the input graph $G$ and $r$ where $r$ is a parameter. Let $\Delta = C\sqrt{k}\sqrt{p(1-q)}(\log kn)^7(p-q)^{r-1}n^{r-1}$, where $C > 0$ is some universal constant. We set $B = A - q \cdot J$ where $J = 1^{n \times n}$. Let $W = V$.

We choose an arbitrary vertex $v$ in $W$ and put $v$ into a sub-cluster $C_i$ where $i \in [k]$. For each node $u$ in $W$, if $\|B_u^r - B_v^r\| \leq \Delta$, then we add $u$ to $C_i$. Next, we remove $C_i$ from $W$. Repeat the above process until $W$ is empty. Then we return all the clusters $C_i$'s.

▶ **Theorem 16.** *Let $p, q \leq 0.75$ be parameters such that $\max\{p(1-p), q(1-q)\} \geq C_0 \log n/n$ where $C_0 > 0$ is some constant. Suppose that $\frac{p-q}{\sqrt{p}} \geq (C_0^2+1)k^{\frac{1}{2}} n^{-\frac{1}{2}+\frac{1}{2(r-1)}} (\log kn)^7$. Let $G$ be a random graph generated from SBM$(n, p, q, k)$ and $r \in [3, O(\log n)]$, then with probability at least $1 - O(n^{-1})$ the algorithm* PowerIteration *can output $k$ hidden clusters for suitable $\Delta$ and $r$.*

## 5.2 The MPC Algorithm

In this section, we show how to implement PowerIteration in the $s$-space MPC model. The pseudocode is found in Algorithm 3. Given a matrix $A$, we use $A_i^{2r}$ to denote the $i$-th row of $A^{2r}$. We use $A_{\cdot j}^{2r}$ to denote the $j$-th column of $A^{2r}$.

**Algorithm 3** MPC-PowerIteration.

**Input:** A SBM graph $G$, $\Delta$;

Let $A$ be an adjacent matrix of $G$, $r$ be the parameter
Let $\Delta = C\sqrt{k}\sqrt{p(1-q)}(\log kn)^7(p-q)^{r-1}n^{r-1}$, where $C \in \mathbb{R}_*^+$
$B = A - q \cdot J$ where $J = 1^{n\times n}$
Let $i = 1$ and $W = V$
Initially, all nodes in $W$ are active
**while** IsActive$(W)$ is true **do**
  Choose an arbitrary vertex $v \in W$ and send it to all machines
  Label $v$ with $i$, i.e., $C_i = \{v\}$
  **for** each machine holding active vertex $u \in W$, we execute the following procedure in parallel **do**
    ComputeNorm$(B, u, v, r)$
    **if** $\|B_u^r - B_v^r\| \leq \Delta$ **then**
      Label $u$ with $i$
  Set nodes in $C_i$ inactive
  $i = i + 1$
Return all the sub-clusters $C_i$'s.

We use IsActive$(W)$ to determine whether there are active nodes in $W$ or not, which can be done in $O(\log_s n)$ rounds. The details of IsActive$(W)$ is given in the full version.

We then use another subroutine ComputeNorm$(B, i, j, r)$ to compute $\|B_i^r - B_j^r\|$. Notice that we can't directly calculate matrix multiplication, which will take lots of rounds, we notice some good properties of $\|B_i^r - B_j^r\|$ and have the following theorem.

▶ **Theorem 17.** *For a fixed $i$ and $j \in [n]$ and any integer $r < O(\log n)$, the subrountine* ComputeNorm$(B, i, j, r)$ *for computing $\|B_i^r - B_j^r\|$ can be implemented in $O(r \log_s n)$ rounds where each machine has memory $s = \Omega(\log n)$.*

Now we give the ideas of ComputeNorm$(B, i, j, r)$. We find that the expansion of $\|B_i^r - B_j^r\|$ has good properties such that we only need to compute the key terms for these $O(r^2)$ terms and the coefficients have good combinatorial explanations. Then we can use graph algorithms to calculate the results. First we note that

$$||B_i^r - B_j^r||^2 = ||(\mathbf{1}_i^T - \mathbf{1}_j^T)(A - qJ)^r||^2 = (\mathbf{1}_i^T - \mathbf{1}_j^T)(A - qJ)^{2r}(\mathbf{1}_i - \mathbf{1}_j)$$
$$= \mathbf{1}_i^T(A - qJ)^{2r}\mathbf{1}_i - \mathbf{1}_i^T(A - qJ)^{2r}\mathbf{1}_j - \mathbf{1}_j^T(A - qJ)^{2r}\mathbf{1}_i + \mathbf{1}_j^T(A - qJ)^{2r}\mathbf{1}_j,$$

so we only need to calculate $\mathbf{1}_x^T(A - qJ)^{2r}\mathbf{1}_y$ where $x, y \in \{i, j\}$.

Now we have the following lemma about the expanded formula.

▶ **Lemma 18.** *We have*

$$\mathbf{1}_x^T(A - qJ)^{2r}\mathbf{1}_y = \mathbf{1}_x^T A^{2r}\mathbf{1}_y + \sum_{0 \le i_1 \le 2r-1} \sum_{0 \le i_t \le 2r-1} X_{i_1,i_t}(A_x^{i_1})J(A_{\cdot y}^{i_t})$$

*where $X_{i_1,i_t}$ is coefficient only related to $n, q$ and $C_i$ and $C_i$ is the total number of different walks with length $i$ from $n$ vertices.*

Since $r = O(\log n)$, there are $\text{poly}(\log n)$ terms in the right hand side. So we can store all coefficients in $\tilde{O}(n)$ space. Notice that $\mathbf{1}_x^T A^{2r}\mathbf{1}_y$ for all $y \in [n]$ is the $i^{th}$ row of $A^{2r}$, i.e., $A_x^{2r}$. To compute $\mathbf{1}_x^T(A - qJ)^{2r}\mathbf{1}_y$, we split it into computing $C_i$, $A_x^{i_1}JA_{\cdot y}^{i_t}$, and $A_x^{2r}$.

### Compute $C_i$ and $A_x^{i_1}JA_{\cdot y}^{i_t}$

We show how to compute the value of any $C_i$ and $A_x^{i_1}JA_{\cdot y}^{i_t}$. Let $\vec{\mathbf{j}}$ be all ones vector, which is a column of $J$. To compute $A_x^i\vec{\mathbf{j}}$, we propose a simple algorithm, i.e., Algorithm 4 that is described as follows.

**Algorithm 4** AixSum$(G(n), r)$: Calculating $A_x^i\vec{\mathbf{j}}$ for all $x \in [n], i \in [2r]$.

**Input:** A graph $G(n)$,$r$
1: **for** each node $u$ in $G$ **do**
2: $A_{u,0} = 1$
3: let $i = 1$
4: **while** $i \le 2r$ **do**
5: **for** each node $u$ in $G$ **do**
6: $sum_u = 0$
7: **for** each neighbor $v$ of $u$ **do**
8: $sum_u += A_{v,i-1}$
9: $A_{v,i} = sum_u$
10: $i = i + 1$
11: Return $A$, $A_{x,i}$ is $A_x^i\vec{\mathbf{j}}$

▶ **Lemma 19.** *For all $x \in [n], i \in [2r]$, Algorithm 4 outputs $A_x^i\vec{\mathbf{j}}$.*

To compute $C_i$ for any $i \in [n]$, we only need to sum up $A_x^i\vec{\mathbf{j}}$ for all $x, i \in [n]$.

Now, let us see how to implement Algorithm 4 in the $s$-space MPC model. Notice that in default, we use the fact that in the $s$-space MPC model, each vertex can visit its neighbors in $O(\log_s n)$ rounds where each machine has memory of $O(n^\delta)$ by Lemma 5.

▶ **Lemma 20.** *In $s$-space MPC model, for all $i \in [2r]$ and $x \in [n]$, there exists an algorithm that can compute all $A_x^i\vec{\mathbf{j}}$ in $O(r \log_s n)$ rounds,where each machine has memory $s = \Omega(\log_s n)$.*

Notice that $A_x^{i_1}JA_{\cdot y}^{i_t} = A_x^{i_1}\vec{\mathbf{j}}(A_y^{i_t}\vec{\mathbf{j}})$, $C_i = \sum_{x\in[n]} A_x^i\vec{\mathbf{j}}$ and we have computed $A_x^i\vec{\mathbf{j}}$ for all $i \in [2r]$ and $x \in [n]$, we can obtain $A_x^{i_1}JA_{\cdot y}^{i_t}$ in constant rounds. So the main round complexity is only about the calculation of $A_x^i\vec{\mathbf{j}}$ and we have the following corollary.

▶ **Corollary 21.** *In $s$-space MPC model, there exists an algorithm that can compute $A_x^{i_1}JA_{\cdot y}^{i_t}$ and $C_i$ in $O(r \log_s n)$ rounds, where each machine has memory $s = \Omega(\log n)$.*

### Compute $A_x^{2r}$

Note that each entry $a_{x,y}^{2r}$ in $A_x^{2r}$ is exactly the number of walks with step size $r$ from $v_x$ to $v_y$. The naive algorithm of computing $A^{2r}$ is to compute the matrix, but it is resources-consuming. Another idea is to compute $a_{x,y}^{2r}$ for any $x$ and $y$, respectively. If each machine can store all vertices within radius $r$, then we can directly compute all $a_{x,y}^{2r}$ $(x, y \in [n])$. Now, we consider the $s$-space MPC model, i.e., single machines cannot store vertices within radius $r$.

▶ **Lemma 22.** *Let $a_{x,y}^{2r}$ be the number of walks from the vertex $x$ to the vertex $y$ after walks with step size $2r$. There exists a procedure* ComputeArx *that can find $a_{x,y}^{2r}$ after $O(r \log_s n)$ rounds where each machine has memory $s = \Omega(\log n)$.*

By taking the union of different vertices, we can get the following corollary.

▶ **Corollary 23.** *Let $A_i^{2r}$ be set of the numbers of walks from the vertex $i$ to the vertex $j$ where $j \in [1, n]$ after walks with step size $r$. The procedure* ComputeArx *can find $A_i^{2r}$ after $O(r \log_s n)$ rounds where each machine has memory $s = \Omega(\log n)$.*

### Compute $\|B_i^r - B_j^r\|$

After obtaining $A_x^i$ and $A_{\cdot y}^i$, the value of $A_x^i J A_{\cdot y}^i$ is the multiplication of the sums of terms in each of two vectors. And the coefficient of each term is the multiplication of $C_i$, $n$ and $q$. Now, we can prove Theorem 17.

**Proof of Theorem 17.** By Lemma 18, $\mathbf{1}_x^T (A - qJ)^{2r} \mathbf{1}_y$ which consists of at most $O(r^2)$ terms with $C_i$, $A_x^{i_1} J A_{\cdot y}^{i_t}$, and $A_x^{2r}$. Let us see the round complexity of computing it. We take the round complexity of computing one key term as an example. We only need to look at the round complexity of computing $\left(\prod_{l=2}^{t-1} C_{i_l}\right) (A_x^{i_1}) J (A_{\cdot y}^{i_t})$. By Corollary 21, we need $O(i \log_s n)$ rounds to compute any $C_i$ where $i \in [2r]$. We can finish calculating $\left(\prod_{l=2}^{t-1} C_{i_l}\right)$ in $O(r \log_s n)$ rounds. By Lemma 20, we can obtain the result of $(A_x^{i_1}) J (A_{\cdot y}^{i_t})$ within $O(r \log_s n)$ rounds. Notice that there is a special term $\mathbf{1}_x^T A^{2r} \mathbf{1}_y^T$. By Corollary 23, we can find it within $O(r \log_s n)$ rounds. There are some other similar computations, which also take $O(r \log_s n)$ rounds. Recall that there are $O(r^2)$ terms, we deal with it by copying the whole graph for poly$(\log n)$ times and then put these results together. Therefore, it takes $O(r \log_s n)$ rounds to calculate $\mathbf{1}_x^T (A - qJ)^{2r} \mathbf{1}_y$. Therefore, we need $O(r \log_s n)$ rounds to finish the calculation of $\|B_i^r - B_j^r\|$. ◀

By Theorem 17, we can finish the proof of Theorem 1.

**Proof of Theorem 1.** The main idea of MPC-PowerIteration(Algorithm 3) is to fix a node $v_i$ first and calculate $\|B_i^r - B_j^r\|$ for any other node $v_j$ in the same cluster to obtain all nodes in the same cluster. We need to store all simplified coefficients in each round which uses $O(nr^2)$ space. For other operations in the algorithms, $O(m)$ space is enough. So the total space complexity is $\tilde{O}(m + nr^2) = \tilde{O}(m)$.

In the full version, we show how to implement IsActive$(W)$ in $O(\log_s n)$ rounds. By Theorem 17, we can finish $\|B_i^r - B_j^r\|$ within $O(r \log_s n)$ rounds. Therefore, we need $O(r \log_s n)$ rounds to find a cluster and all its nodes. There are $k$ hidden clusters and we execute the above procedure sequentially, so the round complexity is $O(kr \log_s n)$. ◀

Now we show how to use more space to trade off round complexity and give the proof of Theorem 2.

**Proof of Theorem 2.** Recall that in MPC-PowerIteration(Algorithm 3), we sequentially find $k$ clusters, that is the reason why there is a factor $k$ in the round complexity. Now, we execute the process in parallel. First, we randomly sample a set $S_k$ of $\Theta(k \log n)$ nodes. With high probability, for each hidden cluster, we sample $\Theta(\log n)$ nodes in $S_k$. Then, for each node $u \in S_k$, we execute ComputeMatrix$(B, u, v, r)$ for each $v \in V$. If $\|B_u^r - B_v^r\| \leq \Delta$, we put $u$ and $v$ in the same cluster. The space for this step is $\tilde{O}(km + knr^2 \log n)$. Then, we will have $k$ clusters with $\Theta(k \log n)$ labels. We remove duplicated labels by keeping the label with the minimum value among all received labels to get one label vertex for each cluster. Then by using these $k$ label vertices, we can use $\tilde{O}(km + knr^2 \log n) = \tilde{O}(km)$ space to cluster all vertices. So, we can find all $k$ clusters in $O(r \log_s n)$ rounds with high probability. ◀

## References


1. Emmanuel Abbe. Community detection and stochastic block models. *Found. Trends Commun. Inf. Theory*, 14(1-2):1–162, 2018.
2. Emmanuel Abbe, Afonso S. Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Trans. Inf. Theory*, 62(1):471–487, 2016.
3. Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
4. MohammadHossein Bateni, Hossein Esfandiari, Manuela Fischer, and Vahab S. Mirrokni. Extreme k-center clustering. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 3941–3949. AAAI Press, 2021.
5. Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA – June 22 – 27, 2013*, pages 273–284. ACM, 2013.
6. Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M Karp, and Jara Uitto. Massively parallel computation of matching and mis in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 481–490, 2019.
7. Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Almost 3-approximate correlation clustering in constant rounds. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 720–731. IEEE, 2022.
8. Aditya Bhaskara and Maheshakya Wijewardena. Distributed clustering via LSH based data partitioning. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 569–578. PMLR, 2018.
9. Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317. ACM, 2012.
10. Mélanie Cambus, Davin Choo, Havu Miikonen, and Jara Uitto. Massively parallel correlation clustering in bounded arboricity graphs. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

11 Flavio Chierichetti, Nilesh N. Dalvi, and Ravi Kumar. Correlation clustering in MapReduce. In *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining*, pages 641–650. ACM, 2014.

12 Vincent Cohen-Addad, Adrian Kosowski, Frederik Mallmann-Trenn, and David Saulpic. On the power of louvain in the stochastic block model. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

13 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021.

14 Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Parallel and efficient hierarchical k-median clustering. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 20333–20345, 2021.

15 Vincent Cohen-Addad, Frederik Mallmann-Trenn, and David Saulpic. Community recovery in the degree-heterogeneous stochastic block model. In Po-Ling Loh and Maxim Raginsky, editors, *Conference on Learning Theory, 2-5 July 2022, London, UK*, volume 178 of *Proceedings of Machine Learning Research*, pages 1662–1692. PMLR, 2022.

16 Vincent Cohen-Addad, Frederik Mallmann-Trenn, and David Saulpic. A massively parallel modularity-maximizing algorithm with provable guarantees. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 – 29, 2022*, pages 356–365. ACM, 2022.

17 Vincent Cohen-Addad, Vahab S. Mirrokni, and Peilin Zhong. Massively parallel k-means clustering for perturbation resilient instances. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4180–4201. PMLR, 2022.

18 Sam Coy, Artur Czumaj, and Gopinath Mishra. On parallel k-center clustering. *arXiv preprint arXiv:2304.05883*, 2023.

19 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

20 Alessandro Epasto, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Scalable diversity maximization via small-size composable core-sets (brief announcement). In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 41–42. ACM, 2019.

21 Manuela Fischer and Andreas Noever. Tight analysis of parallel randomized greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020.

22 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation – 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, volume 7074 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011.

23 Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In Paulo Ferreira, Thomas R. Gross, and Luís Veiga, editors, *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pages 59–72. ACM, 2007.

**24** Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010.

**25** Chandra Sekhar Mukherjee and Jiapeng Zhang. Detecting hidden communities by power iterations with connections to vanilla spectral algorithms, 2022. doi:10.48550/ARXIV.2211.03939.

**26** Xinghao Pan, Dimitris S. Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. Parallel correlation clustering on big graphs. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 82–90, 2015.

**27** Peng Wang, Zirui Zhou, and Anthony Man-Cho So. A nearly-linear time algorithm for exact community recovery in stochastic block model. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 10126–10135. PMLR, 2020.

**28** Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

**29** Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In Erich M. Nahum and Dongyan Xu, editors, *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010*. USENIX Association, 2010.

# Connectivity in the Presence of an Opponent

**Zihui Liang**[1]
University of Electronic Science and Technology of China, Chengdu, China

**Bakh Khoussainov**[1]
University of Electronic Science and Technology of China, Chengdu, China

**Toru Takisaka**
University of Electronic Science and Technology of China, Chengdu, China

**Mingyu Xiao**
University of Electronic Science and Technology of China, Chengdu, China

## Abstract

The paper introduces two player connectivity games played on finite bipartite graphs. Algorithms that solve these connectivity games can be used as subroutines for solving Müller games. Müller games constitute a well established class of games in model checking and verification. In connectivity games, the objective of one of the players is to visit every node of the game graph infinitely often. The first contribution of this paper is our proof that solving connectivity games can be reduced to the incremental strongly connected component maintenance (ISCCM) problem, an important problem in graph algorithms and data structures. The second contribution is that we non-trivially adapt two known algorithms for the ISCCM problem to provide two efficient algorithms that solve the connectivity games problem. Finally, based on the techniques developed, we recast Horn's polynomial time algorithm that solves explicitly given Müller games and provide the first correctness proof of the algorithm. Our algorithms are more efficient than that of Horn's algorithm. Our solution for connectivity games is used as a subroutine in the algorithm.

**2012 ACM Subject Classification** Theory of computation → Logic and verification; Theory of computation → Complexity theory and logic; Theory of computation → Dynamic graph algorithms

**Keywords and phrases** Explicit Müller games, games played on finite graphs, winning strategies, synthesis and analysis of games

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.79

**Funding** *Bakh Khoussainov*: Bakh Khoussainov acknowledges the National Science Foundation of China under Grant No.62172077.

## 1 Introduction

### 1.1 Müller games given explicitly

In the area of logic, model checking, and verification of reactive systems, studying games played on graphs is a key research topic [10]. This is mostly motivated through modelling reactive systems and reductions of model checking problems to games on graphs. Understanding the algorithmic content of determinacy results is also at the core of this research. Müller games constitute a well-established class of games for verification. Recall that a *Müller game* $\mathcal{G}$ is a tuple $(V_0, V_1, E, \Omega)$, where

- The tuple $G = (V_0 \cup V_1, E)$ is a finite directed bipartite graph so that $V_0$ and $V_1$ partition the set $V = V_0 \cup V_1$. Usually $G$ is called the arena of $\mathcal{G}$.
- The set $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$ of edges.

[1] Corresponding authors.

- $V_0$ and $V_1$ are sets from which player 0 and player 1, respectively, move. Positions in $V_\sigma$ are called player $\sigma$ positions, $\sigma \in \{0,1\}$.
- $\Omega \subseteq 2^V$ is a collection of winning sets.

Say that the *game* $\mathcal{G} = (V_0, V_1, E, \Omega)$ is *explicitly given* if $V$, $E$, and all sets in $\Omega$ are fully presented as input. The (input) size of explicitly given Müller game is thus bounded by $|V| + |E| + 2^{|V|} \cdot |V|$. Finally, the *game graph* of the Müller game $\mathcal{G}$ is the underlying bipartite graph $G = (V_0 \cup V_1, E)$.

For each $v \in V$, let $E(v) = \{u \mid E(v,u)\}$ be the set of successors of $v$. Let $X \subseteq V$. Call the set $E(X) = \bigcup_{v \in X} E(v)$ the successor of $X$. Similarly, for a $v \in V$, the predecessor of $v$ is the set $E^{-1}(v) = \{u \mid (u,v) \in E\}$. Call the set $E^{-1}(X) = \bigcup_{v \in X} E^{-1}(v)$ the predecessor of $X$.

Let $\mathcal{G} = (V_0, V_1, E, \Omega)$ be a Müller game. The players play the game by moving a given token along the edges of the graph. The token is initially placed on a node $v_0 \in V$. The play proceeds in rounds. At any round of the play, if the token is placed on a player $\sigma$'s node $v$, then player $\sigma$ chooses $u \in E(v)$, moves the token to $u$ and the play continues on to the next round. Formally, a play (starting from $v_0$) is a sequence $\rho = v_0, v_1, \ldots$ such that $v_{i+1} \in E(v_i)$ for all $i \in \mathbb{N}$. If a play reaches a position $v$ such that $E(v) = \emptyset$, then player 1 wins the play. For an infinite play $\rho$, set $\mathsf{Inf}(\rho) = \{v \in V \mid \exists^\omega i (v_i = v)\}$. We say player 0 wins the play $\rho$ if $\mathsf{Inf}(\rho) \in \Omega$; otherwise, player 1 wins the play.

A strategy for player $\sigma$ is a function that takes as inputs initial segments of plays $v_0, v_1, \ldots, v_k$ where $v_k \in V_\sigma$ and output some $v_{k+1} \in E(v_k)$. A strategy for player $\sigma$ is winning from $v_0$ if, assuming player $\sigma$ follows the strategy, all plays starting from $v_0$ generated by the players are winning for player $\sigma$. The game $\mathcal{G}$ is determined if one of the players has a winning strategy. Müller games are Borel games, and hence, by the result of Martin [18], they are determined. Since Müller games are determined we can partition the set $V$ onto two sets $Win_0$ and $Win_1$, where $v \in Win_\sigma$ iff player $\sigma$ wins the game starting at $v$, $\sigma \in \{0,1\}$. To solve a given Müller game $\mathcal{G} = (V_0, V_1, E, \Omega)$ means to find the sets $Win_0$ and $Win_1$. There are several known algorithms that solve Müller games. These algorithms provide the basis for analysis and synthesis of Müller games. In particular, these algorithms extract finite state winning strategies for the players [8, 12, 13, 19, 20, 22]. Also, efficiency of algorithms depend on the underlying structure of graphs [17] [9]. We stress that the algorithms that solve Müller games depend on the presentations of the games. The problem of solving Müller games is typically in PSPACE for many reasonable representations [19, 20]. However, if the winning condition is represented as a Zielonka tree [22] or as the well-known parity condition, then solving the games turns into a $NP \cap co\text{-}NP$ problem [5]. P. Hunter and A. Dawar [13] investigate five other representations: win-set, Muller, Zielonka DAGs, Emerson-Lei, and explicit Muller. They show that the problem of the winner is PSPACE-hard for the first four representations. F. Horn [12] provides a polynomial time algorithm that solves explicit Müller games. However, his proof of correctness has non-trivial flaws. So, we provide an alternative correctness proof based on ideas totally independent of Horn's. Designing new algorithms, improving and analysing the state of the art techniques in this area is a key research direction. This paper contributes to this.

## 1.2 Connectivity games

One motivation for defining connectivity games comes from solving Müller games. Many algorithms that solve Müller games or its variants are recursive. Given a Müller game $\mathcal{G}$, one constructs a set of smaller Müller games. The solution of the games $\mathcal{G}'$ from this set is then used to solve $\mathcal{G}$. Through an iteration process, these reductions produce sequences of the

form $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_r$, where $\mathcal{G}_{i+1} = \mathcal{G}'_i$ such that $\mathcal{G}'_r = \mathcal{G}_{r+1}$. The key point is that solving the game $\mathcal{G}_r$ at the base of this iteration boils down to investigating connectivity of the graph $G_r$ in the game-theoretic setting. Namely, to win the game $\mathcal{G}_r$, one of the players has to visit all the nodes of $\mathcal{G}_r$ infinitely often. This observation calls for deeper and refined analysis of those Müller games $\mathcal{G} = (V_0, V_1, E, \Omega)$ where the objective of player 0 is to visit all the nodes of the underlying graph $G$, that is, $\Omega = \{V\}$. We single out these games:

▶ **Definition 1.** *A Müller game $\mathcal{G} = (V_0, V_1, E, \Omega)$ is called a* **connectivity game** *if $\Omega$ is a singleton that consists of $V$.*

The second motivation to investigate the connectivity games comes from the concept of connectivity itself. The notion of (vertex) connectivity is fundamental in graph theory and its applications. There is a large amount of work ranging from complexity theoretic issues to designing efficient data structures and algorithms that aim to analyse connectivity in graphs. Connectivity in graphs and graph like structures is well-studied in almost all areas of computer science in various settings and motivations. For undirected graphs, connectivity of a graph $G$ is defined through existence of paths between all vertices of $G$. For directed graphs $G$ connectivity is defined through strong connectivity. The digraph $G$ is strongly connected if for any two vertices $x$ and $y$ there exist paths from $x$ to $y$ and from $y$ to $x$. One can extend these notions of (vertex) connectivity into a game-theoretic setting as follows. There are two players: player 0 and player 1. A token is placed on a vertex $v_0$ of a bipartite graph $G = (V_0 \cup V_1, E)$. Player 0 starts the play by moving the token along an outgoing edge $(v_0, v_1)$. Player 1 responds by moving the token along an outgoing edge from the vertex $v_1$, say $(v_1, v_2)$. This continues on and the players produce a path $v_0, v_1, \ldots, v_k$ called a play starting at $v_0$. Say that player 0 wins the play $v_0, v_1, \ldots, v_k$ if the play visits every node in $V$. Call thus described game *forced-connectivity game*. A posssible scenario for this situation is that player 0 wants to pass a message through all the nodes of a given network in the presense of an adversary. If player 0 has a winning strategy, then we say that the player wins the game starting at $v_0$. Winning this forced-connectivity game from $v_0$ does not always guarantee that the player wins the game starting at any other vertex. Therefore we can define game-theoretic connectivity as follows. A directed bipartite graph $G$ is *forced-connected* if player 0 wins the forced-connectivity game in $G$ starting at any vertex of $G$. Thus, finding out if $G$ is a forced-connected is equivalent to solving connectivity games as in Definition 1.

▶ **Definition 2.** *Let $\mathcal{G} = (V_0, V_1, E, \Omega)$ be a connectivity game. Call the bipartite graph $G = (V_0, V_1, E)$* **forced-connected** *if player 0 wins the game $\mathcal{G}$.*

The third motivation is related to generalised Büchi winning condition. The generalised Büchi winning condition is given by subsets $F_1$, ..., $F_k$ of the game graph $G$. Player 0 wins a play if the play meets each of these winning sets $F_1$, ..., $F_k$ infinitely often. Our connectivity games winning condition can be viewed as a specific generalised Büchi winning condition where the accepting sets are all singletons.

## 1.3 Our contributions

The focus of this paper is two-fold. On the one hand, we study connectivity games and provide the state-of-the-art algorithms for solving them. H. Bodlaender, M. Dinneen, and B. Khoussainov [3, 4] call connectivity games *update games*. Their motivation comes from modelling the scenario where messages should be passed to all the nodes of the network in the presence of adversary. On the other hand, using the connectivity game solution process as a subroutine, we recast Horn's polynomial time algorithm that solves explicitly given Müller games and provide a proof of its correctness. We detail these below.

1. Our first contribution is that given a connectivity game $\mathcal{G}$, we construct a sequence of directed graphs $\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_s$ such that player 0 wins $\mathcal{G}$ if and only if $\mathcal{G}_s$ is strongly connected [See Theorem 8]. Due to this result, we reduce solving connectivity game problem to the incremental strongly connected component maintenance (ISCCM) problem, one of the key problems in graph algorithms and data structure analysis [1, 11].
2. A standard brute-force algorithm that solves the connectivity game $\mathcal{G}$ runs in time $\mathbf{O}(|V|^2(|V| + |E|))$. H. Bodlaender, M. Dinneen, and B. Khoussainov in [3, 4, 7, 16] provided algorithms that solve the connectivity games in $\mathbf{O}(|V||E|)$. Due to Theorem 8, we solve the connectivity game problem by adapting two known algorithms that solve the ISCCM problem. The first algorithm is by Haeupler et al. [11] who designed the *soft-threshold search algorithm* that handles sparse graphs. Their algorithm runs in time $\mathbf{O}(\sqrt{m}m)$, where $m$ is the number of edges. The second is the solution by Bender et al. [1, 2]. Their algorithm is best suited for the class of dense graphs and runs in time of $\mathbf{O}(n^2 \log n)$, where $n$ is the number of vertices. By adapting these algorithms, we design new algorithms to solve the connectivity games. The first algorithm, given a connectivity game $\mathcal{G}$, runs in time $\mathbf{O}((\sqrt{|V_1|}+1)|E| + |V_1|^2)$ [See Theorem 9]. The first feature of this algorithm is that the algorithm solves the problem in linear time in $|V_0|$ if $|V_1|$ is considered as a parameter. The parameter constant in this case is $|V_1|^{3/2}$. The second feature is that the algorithm runs in linear time if the underlying game graph is sparse. Our second algorithm solves the connectivity game in time $\mathbf{O}((|V_1| + |V_0|) \cdot |V_0| \log |V_0|)$ [See Theorem 10]. In contrast to the previous algorithm, this algorithm solves the connectivity game problem in linear time in $|V_1|$ if $|V_0|$ is considered as a parameter. The parameterised constant is $|V_0| \log |V_0|$. Furthermore, the second algorithm is more efficient than the first one on dense graphs. These two algorithms outperform the standard bound $O(|V||E|)$, mentioned above, for solving the connectivity games. As a framework, this is similar to the work of K. Chatterjee and M. Henzinger [6] who improved the standard $O(|V||E|)$ time bound for solving Büchi games to $O(|V|^2)$ bound through analysis of maximal end-component decomposition algorithms.
3. In [12] Horn provided a polynomial time algorithm that solves explicitly given Müller games. In his algorithm, Horn uses the standard procedure of solving connectivity games as a subroutine. Directly using our algorithms above, as a subroutine to Horn's algorithm, we obviously improve Horn's algorithm in an order of magnitude. Horn's proof of correctness uses three lemmas (see Lemmas 5, 6, and 7 in [12]). However, his Lemmas 6 and 7 contain non-trivial flaws. We provide our independent proof of correctness. In terms of ideas, our proof is completely different from Horn's proof ideas. We discuss these in Section 6. To the best of our knowledge, this is the first work that correctly and fully recasts Horn's polynomial time algorithm with the efficient sub-routine for solving the connectivity games. Furthermore, in terms of running time, our algorithms perform better than that of Horn's algorithm [See Theorem 20 and Theorem 21]. For instance, one of our algorithms decreases the degree of $|\Omega|$ from $|\Omega|^3$ in Horn's algorithm to $|\Omega|^2$ [See Theorem 21]. Since $|\Omega|$ is bounded by $2^{|V|}$, the improvement is significant.

## 2 A characterization theorem

A Müller game $\mathcal{G} = (V_0, V_1, E, \Omega)$ is a connectivity game if $\Omega = \{V\}$. In this section we focus on connectivity games $\mathcal{G}$. In the study of Müller games, often it is required that for each $v$ the successor set $E(v) = \{u \mid (v, u) \in E\}$ is not empty. We do not put this condition as it will be convenient for our analysis of connectivity games to consider cases

when $E(v) = \emptyset$. Recall that a strongly connected component of a directed graph is a maximal set $X$ such that there exists a path between any two vertices of $X$. Denote the collection of all strongly connected components of the game graph $G$ of game $\mathcal{G}$ by $SCC(\mathcal{G})$. For all distinct components $X, Y \in SCC(\mathcal{G})$, we have $X \cap Y = \emptyset$ and $\bigcup_{X \in SCC(\mathcal{G})} X = V$.

▶ **Definition 3.** *Let $\mathcal{G}$ be a connectivity game. Consider two sets $U \subseteq V_1$ and $S \subseteq V$. Define*

$$Force(U, S) = \{v \mid v \in (E^{-1}(S) \setminus S) \cap U \text{ and } E(v) \subseteq S\}.$$

▶ **Definition 4.** *We say that a set $X \subseteq V$ in game $\mathcal{G}$ is a* **forced trap (FT)** *if either $|X| = 1$ or if $|X| > 1$ then $E(X \cap V_1) \subseteq X$ and $X$ is strongly connected. Further $X$ is* **forced-connected component (FCC)** *if either $|X| = 1$ or if $|X| > 1$ then the sub-game $\mathcal{G}(X)$ of the game $\mathcal{G}$ played in $X$ is forced-connected.*

▶ **Lemma 5.** *Let $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where $k > 1$, be a collection of FTs that partition the game graph $G$. If $\mathcal{G}$ is forced-connected then for every $X \in \mathcal{C}$ there is a $Y \in \mathcal{C}$ distinct from $X$ such that either $Y$ is a singleton consisting of a player 1's node and $E(Y) \subseteq X$, or $Y$ has player 0's node $y$ with $E(y) \cap X \neq \emptyset$.*

We now define the sequence $\{\mathcal{G}_k\}_{k \geq 0}$ of graphs. Initially, in $G_0$, the edges are only those that start in the nodes of player 0. Then the inductive construction increases the set of edges as follows. Intuitive explanation of the process is the following. Throughout the sequence, the invariant that FTs coincide with strongly connected components is maintained. By Lemma 7, these FTs are FCCs. This is ensured at each iteration by only adding the edges of those Player 1 vertices from where Player 1 is forced onto an existing SCC in the graph constructed so far. When this iteration terminates, the FCCs of the original graph coincide with the FCCs of the resulting graph, which in turn coincides with the SCCs by the invariant.

Here is now a formal process. We will call each $\mathcal{G}_k$ the $k^{th}$-derivative of $\mathcal{G}$. We will also view each $\mathcal{G}_k$ as a connectivity game. Our construction is the following.

- Initially, for $k = 0$, set $F_0 = \emptyset$, $U_0 = V_1$ and $\mathcal{G}_0 = (V_0, V_1, E_0)$, where $E_0$ consists of all outgoing edges in $E$ of player 0.
- For $k > 0$, consider the set $F_k = \bigcup_{S \in SCC(\mathcal{G}_{k-1})} Force(U_{i-1}, S)$, and define $U_k = U_{k-1} \setminus F_k$, $\mathcal{G}_k = (V_0, V_1, E_k)$, where $E_k = E_{k-1} \cup \{(v, u) \mid v \in F_k \text{ and } (v, u) \in E\}$.

Note that the SCCs of $\mathcal{G}_0$ are all singletons. For $k = 1$ we have the following. The set $F_1$ consists of all player 1 nodes of out-degree 1. The set $E_1$ contains $E_0$ and all outgoing edges from the set $F_1$. We note that each SCC of $\mathcal{G}_1$ is also a FT in $\mathcal{G}_1$. Therefore each SCC $X$ in $\mathcal{G}_1$ is also a maximal FT. Observe that each $F_k \subseteq U_{k-1}$ consists of player 1's nodes $v$ such that all moves of player 1 from $v$ go to the same SCC in $\mathcal{G}_{k-1}$. Moreover, $U_k$ is the set of player 1's nodes whose outgoing edges aren't in $E_k$. Now we list some properties of the sequence $\{\mathcal{G}_k\}_{k \geq 0}$. A verification of these properties follows from the construction:

- For every player 1's node $v$ and $k > 0$, the outgoing edges of $v$ are in $E_k \setminus E_{k-1}$ iff all the outgoing edges of $v$ point to the same SCC in $\mathcal{G}_{k-1}$ and in $\mathcal{G}_{k-1}$ the out-degree of $v$ is 0.
- For each $k \geq 0$, every SCC in $\mathcal{G}_k$ is a FT in $\mathcal{G}_k$.
- For all $k \geq 0$ we have $F_{k+1} \subseteq U_k \subseteq U_{k-1} \subseteq \ldots \subseteq U_0 = V_1$.
- For all $k_1 \neq k_2$ we have $F_{k_1} \cap F_{k_2} = \emptyset$.
- If $F_k = \emptyset$ with $k > 0$ then for all $i \geq k$, $\mathcal{G}_i = \mathcal{G}_{k-1}$. We call the minimal such $k$ the **stabilization point** and denote it by $s$. Note that $s \leq |V_1|$.
- If for all $X \in SCC(\mathcal{G}_k)$, either $|X| > 1$ or $X$ is a singleton consisting of player 0's node only then $\mathcal{G}_k = \mathcal{G}$.
- For each $k \geq 0$ and player 1's node $v$, if $v$ is in a nontrivial SCC in $\mathcal{G}_k$ then all $v$'s outgoing edges from $v$ are in $E_k$.

▶ **Lemma 6.** *If $\mathcal{G}$ is forced-connected and $|SCC(\mathcal{G}_k)| > 1$, then $\mathcal{G}_k \neq \mathcal{G}_{k+1}$.*

Given a connectivity game $\mathcal{G}$, we now construct a sequence of forests $\{\Gamma_k(\mathcal{G})\}_{k\geq 0}$ by induction. The idea is to represent the interactions of the SCCs of the graphs $\mathcal{G}_k$ with SCCs of the $\mathcal{G}_{k-1}$, for $k = 1, 2, \ldots$. The sequence of forests $\Gamma_k(\mathcal{G}) = (N_k, Son_k)$, $k = 0, 1, \ldots$, is defined as follows:

- For $k = 0$, set $\Gamma_0(\mathcal{G}) = (N_0, Son_0)$, where $N_0 = \{\{v\} \mid v \in V\}$ and $Son_0(\{v\}) = \emptyset$ for all $v \in V$.
- For $k > 0$, let $C = SCC(\mathcal{G}_k) \setminus N_{k-1}$ be the set of new SCCs in $\mathcal{G}_k$. Define the forest $\Gamma_k(\mathcal{G}) = (N_k, Son_k)$, where
  1. $N_k = N_{k-1} \cup C$, and
  2. $Son_k = Son_{k-1} \cup \{(X, Y) \mid X \in C, Y \in SCC(\mathcal{G}_{k-1}) \text{ and } Y \subset X\}$.

  Thus the new SCCs $X$ that belong to $C$ have become the roots of the trees in the forest $\Gamma_k(\mathcal{G})$. The children of $X$ are now SCCs in $\mathcal{G}_{k-1}$ that are contained in $X$.

Note that if $s$ is the stabilization point of the sequence $\{\mathcal{G}_k\}_{k\geq 0}$, then for all $k \geq s$ we have $\Gamma_k(\mathcal{G}) = \Gamma_s(\mathcal{G})$. Therefore, we set $\Gamma(\mathcal{G}) = \Gamma_s(\mathcal{G})$. Thus, for the forest $\Gamma(\mathcal{G})$ we have $N = N_s$ and $Son = Son_s$. The following properties of the forest $\Gamma(\mathcal{G})$ can easily be verified:

- For all nodes $X \in N$, $X$'s sons partition $X = \bigcup_{Y \in Son(X)} Y$.
- For all nodes $X \in N$ with $|X| > 1$, $E(X \cap V_1) \subseteq X$.
- The roots of $\Gamma(\mathcal{G})$ are strongly connected components of $\mathcal{G}_s$.

▶ **Lemma 7.** *Consider $\Gamma(\mathcal{G}) = (N, Son)$. Let $X \in N$ be such that $|X| > 1$. Then the sub-game $\mathcal{G}(X)$ of the game $\mathcal{G}$ played in $X$ is forced-connected.*

Given the results above, we now relate solving forced connectivity problem to strong connectedness in directed graphs:

▶ **Theorem 8** (Characterization Theorem)**.** *The connectivity game $\mathcal{G}$ is forced-connected if and only if the directed graph $\mathcal{G}_s$ is strongly connected.*

## 3 Solving connectivity games efficiently

In a dynamic setting the increment strongly connected maintenance (ISCCM) problem is stated as follows. Initially, we are given $n$ vertices and the empty edge set. A sequence of edges $e_1, \ldots, e_m$ are added. No multiple edges and loops are allowed. The goal is to design a data structure that maintains the SCCs of the graphs after each addition of edges. By Theorem 8 the connectivity games problem is reduced to the incremental strongly connected component maintenance (ISCCM) problem. Note that Tarjan's algorithm solves the static version of the strongly connected component maintenance problem in time $\mathbf{O}(m)$ [21].

We mention two algorithms that solve the ISCCM problem. The first is the *soft-threshold search algorithm* by Haeupler et al. [11] that handles sparse graphs. Their algorithm runs in time $\mathbf{O}(\sqrt{m}m)$. The second is by Bender et al. [1, 2]. Their algorithm is best suited for the class of dense graphs and runs in time of $\mathbf{O}(n^2 \log n)$. We adapt these algorithms carefully in the proofs of our Theorems 9 and 10 below.

▶ **Theorem 9.** *The connectivity game $\mathcal{G}$ can be solved in time $\mathbf{O}((\sqrt{|V_1|}+1)|E| + |V_1|^2)$.*

We point out two features of this theorem. The first is that if the cardinality $|V_1|$ is considered as a parameter, then we can solve the problem in linear time in $|V_0|$. The parameter constant in this case is $|V_1|^{3/2}$. The second feature is that the algorithm runs in linear time if the underlying game graph is sparse. Our second theorem is the following:

▶ **Theorem 10.** *The connectivity game $\mathcal{G}$ can be solved in time* $\mathbf{O}((|V_1| + |V_0|) \cdot |V_0| \log |V_0|)$.

In comparison to the theorem above, this theorem implies that we can solve the problem in linear time in $|V_1|$. The parameterised constant is $|V_0| \log |V_0|$. Furthermore, the algorithm is more efficient than the first one on dense graphs.

Finally, both of the algorithms outperform the standard known bound $\mathbf{O}(|V||E|)$ that solves the connectivity games.

## 4 Solving explicitly given Müller games

We start with standard notions about games on graphs. Let $\mathcal{G}$ be a Müller game. A set $S \subseteq V$ determines a subgame in $\mathcal{G}$ if for all $v \in S$ we have $E(v) \cap S \neq \emptyset$. We call $\mathcal{G}(S)$, the subgame of $\mathcal{G}$ determined by $S$. The set $S \subseteq V$ is a $\sigma$-trap in $G$ if $S$ determines a subgame in $\mathcal{G}$ and $E(S \cap V_\sigma) \subseteq S$.

Let $Win_\sigma(\mathcal{G})$ be the set of all $v$ in $\mathcal{G}$ such that player $\sigma$ wins $\mathcal{G}$ starting from $v$. If $Win_\sigma(\mathcal{G}) = V$, we say that player $\sigma$ wins $\mathcal{G}$. Otherwise, we say that player $\sigma$ cannot win $\mathcal{G}$.

Let $Attr_\sigma(X, G(Y))$ be the set of all $v$ in $Y$ such that player $\sigma$ can force the token from $v$ to $X$ in game $\mathcal{G}(Y)$.

Let $\Omega$ be the set of all winning sets of the Müller game $\mathcal{G}$. We *topologically order* $<$ the set $\Omega$, that is, for all distinct $X, Y \in \Omega$, if $X \subsetneq Y$ then $X < Y$. Thus, if $W_1 < W_2 < \ldots < W_s$ is a topological linear order on $\Omega$ then we have this. If $i < j$ then $W_i \not\supseteq W_j$.

Below we provide several results that are interesting on their own. We will also use them in our analysis of Müller games.

▶ **Lemma 11.** *Let $\mathcal{G}$ be a game, $F_0 = \Omega$ and $F_1 = 2^V \setminus \Omega$. If $V \in F_\sigma$ and for all $v \in V$, either $Attr_\sigma(\{v\}, G) = V$ or player $\sigma$ wins $\mathcal{G}(V \setminus Attr_\sigma(\{v\}, G))$ then player $\sigma$ wins $\mathcal{G}$.*

The proof of the next lemma uses the lemma above:

▶ **Lemma 12.** *Let $\mathcal{S} = \{S_1, S_2, \ldots, S_k\} \subseteq 2^V \setminus \{V\}$ be the collection of all 0-traps in $\mathcal{G}$ and assume that $V \in \Omega$. If for all $S_i \in \mathcal{S}$, player 1 can't win $\mathcal{G}(S_i)$ then player 0 wins $\mathcal{G}$.*

The next lemma shows that we can reduce the size of the wining condition set $\Omega'$ if one of the sets $W \in \Omega'$ is minimal (with respect to $\subseteq$) and not forced-connected. The proof uses Lemmas 11 and 12.

▶ **Lemma 13.** *Let $W \subseteq V$ be a subgame. If $\mathcal{G}(W)$ isn't forced-connected and no winning set in $\Omega$ is contained in $W$, then $Win_1(\mathcal{G}) = Win_1(\mathcal{G}')$, where $\mathcal{G}'$ is the same as $\mathcal{G}$ but has the additional winning set: $\Omega' = \Omega \cup \{W\}$.*

Let $\mathcal{G}$ be Müller game with $\Omega = \{W_1, W_2, \ldots, W_s\}$. For the next two lemmas and the follow-up theorem we assume that there exists a $W \in \Omega$ such that $\mathcal{G}(W)$ is forced-connected and $W$ isn't a 1-trap. The following is a construction that occurs naturally if one wants to analyse Müller games. We attribute this to Horn [12]:

▶ **Definition 14** (Horn's construction). *Let $W \in \Omega$ such that $\mathcal{G}(W)$ is forced-connected and is not a 1-trap. The game $\mathcal{G}_W = (G_W, \Omega_W)$ determined by $W$ is defined as follows:*

1. $V_W = V_0 \cup V_1 \cup \{\mathbf{W}\}$, *where* $\mathbf{W}$ *is a player 1's new vertex.*
2. $E_W = E \cup (V_0 \cap W) \times \{\mathbf{W}\} \ \cup \ \{\mathbf{W}\} \times (E(V_1 \cap W) \setminus W)$.
3. $\Omega_W = (\Omega \cup \{W' \cup \{\mathbf{W}\} \mid W' \in R\}) \setminus (R \cup \{W\})$, *where the set $R$ is the following* $R = \{W' \mid W' \in \Omega$ *and* $W \subset W'\}$.

Note that $|\Omega_W| + 1 = |\Omega|$, and $\mathcal{G}_W(W)$ is forced-connected. Thus, similar to the lemma above, Horn's construction also reduces the size of $\Omega$. Now our goal is to show that Horn's construction preserves the winners of the original game. This is shown in the next two lemmas. Here we note that Horn's original proof of his correctness followed a different line of proof; this will be explained later.

▶ **Lemma 15.** *We have* $Win_0(\mathcal{G}_W) \setminus \{\mathbf{W}\} \subseteq Win_0(\mathcal{G})$.

**Proof.** Let $\sigma_W$ be a winning strategy for player 0 in game $\mathcal{G}_W$ starting at $s \in V$. We now describe a winning strategy for player 0 in $\mathcal{G}$ starting from $s$. Player 0 plays the game $\mathcal{G}$ by simulating plays $\rho$ consistent with $\sigma_W$ in $\mathcal{G}_W$. If a play $\rho$ stays out of $\mathbf{W}$, then the player 0 copies $\rho$ in $\mathcal{G}$. Once $\rho$ moves to $\mathbf{W}$, then player 0 in $\mathcal{G}$ moves to any node in $W \cap V_1$. Then player 0 stays in $W$ and uses its strategy to visit every node in $W$. If player 1 moves out of $W$ to a node $u$ in $\mathcal{G}$, this will correspond to a move by player 1 from $\mathbf{W}$ to $u$ in $\mathcal{G}_W$. Player 0 continues on simulating $\rho$.

Let $\rho'$ be the play in $\mathcal{G}$ consistent with the strategy. If $\rho$ meets $\mathbf{W}$ finitely often then $\mathsf{Inf}(\rho) = \mathsf{Inf}(\rho')$ and $\mathsf{Inf}(\rho') \in \Omega$. If $\rho$ never moves out of $\mathbf{W}$ from some point on, then $\mathsf{Inf}(\rho') = W$. In both cases player 0 wins. If the simulation leaves $\mathbf{W}$ infinitely often, then $\mathsf{Inf}(\rho) \in \{W' \cup \{\mathbf{W}\} \mid W' \in R\}$ and $W \subseteq \mathsf{Inf}(\rho)$. Therefore

$$\mathsf{Inf}(\rho') \subseteq \mathsf{Inf}(\rho) \setminus \{\mathbf{W}\} \cup W = \mathsf{Inf}(\rho) \setminus \{\mathbf{W}\} \subseteq \mathsf{Inf}(\rho'),$$

and hence $\mathsf{Inf}(\rho') = \mathsf{Inf}(\rho) \setminus \{\mathbf{W}\} \in R$, and player 0 wins. ◀

The next lemma is more involved. Assume that the set $W' \subseteq V$ determines a subgame in $\mathcal{G}$. Then $W'$ also determines a subgame of $\mathcal{G}_W$. We call the set $W'$ *extendible* if $W' \cup \{\mathbf{W}\}$ is a subgame of $\mathcal{G}_W$. Note that there could exist non-extendible $W'$. In particular, some winning sets in $\Omega$ could become non-extendible in $\mathcal{G}_W$. In the analysis of $Win_1(\mathcal{G}_W)$ extendible and non-extendible sets must be taken into account. The lemma below does extactly that.

▶ **Lemma 16.** *We have* $Win_1(\mathcal{G}_W) \setminus \{\mathbf{W}\} \subseteq Win_1(\mathcal{G})$.

**Proof.** We define the following two sets of subgames of the game $\mathcal{G}$. The first set $\mathcal{A}$ is the following set of subgames of $\mathcal{G}$:

$$\{W' \mid W' \text{ is extendible \& player 1 wins } \mathcal{G}_W(W' \cup \{\mathbf{W}\})\}.$$

Note that if $W' \in \mathcal{A}$ then player 1 wins the subgame $\mathcal{G}_W(W')$. The second set $\mathcal{B}$ is the following set of subgames of $\mathcal{G}$:

$$\{W' \mid W \not\subseteq W' \text{ and player 1 wins } \mathcal{G}_W(W')\}.$$

Now we define the set $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$. The sets $\mathcal{A}$ and $\mathcal{B}$ are disjoint. The set $W$ does not belong to $\mathcal{S}$ because $W \cup \{\mathbf{W}\}$ is not a subgame in $\mathcal{G}_W$ and $W \notin \mathcal{B}$ by definition of $\mathcal{B}$. Note that the set $\mathcal{B}$ can contain sets $W'$ that are subsets of non-extendible (winning) sets.

Since player 1 wins $\mathcal{G}_W(Win_1(\mathcal{G}_W))$, $Win_1(\mathcal{G}_W)$ is a 0-trap in $\mathcal{G}_W$ and it's easy to see that $Win_1(\mathcal{G}_W) \setminus \{\mathbf{W}\}$ is also a 0-trap in $\mathcal{G}$. Then if $\mathbf{W} \in Win_1(\mathcal{G}_W)$ then $Win_1(\mathcal{G}_W) \setminus \{\mathbf{W}\} \in \mathcal{A}$, otherwise $Win_1(\mathcal{G}_W) \setminus \{\mathbf{W}\} \in \mathcal{B}$. Since $Win_1(\mathcal{G}_W) \setminus \{\mathbf{W}\} \in \mathcal{S}$, to prove the lemma it suffices to show that player 1 wins $\mathcal{G}(S)$ for all $S \in \mathcal{S}$.

Topologically order $\mathcal{S}$: $S_1 < S_2 < \ldots < S_s$. For each $\ell = 1, 2, \ldots, s$, we want to show that player 1 wins $\mathcal{G}(S_\ell)$. As player 1 wins $\mathcal{G}_W(S_\ell)$, for all 1-traps $S' \subset S_\ell$ player 1 wins $\mathcal{G}_W(S')$. Let $\mathcal{T} = \{T_1, T_2, \ldots T_t\} \subseteq 2^{S_\ell} \setminus \{S_\ell\}$ be all 1-traps in the game $\mathcal{G}(S_\ell)$. For each $T_i \in \mathcal{T}$ we reason as follows.

*Case 1*: $W \subseteq T_i$. Then $E_W(\mathbf{W}) \cap T_i = (E_W(V_1 \cap W) \setminus W) \cap T_i = (E_W(V_1 \cap W) \setminus W) \cap S_\ell = E_W(\mathbf{W}) \cap S_\ell$. Since $W \subseteq S_\ell$ implies player 1 wins $\mathcal{G}_W(S_\ell \cup \{\mathbf{W}\})$ and $E_W(\mathbf{W}) \cap S_\ell \neq \emptyset$, $T_i \cup \{\mathbf{W}\}$ is also a 1-trap in the game $\mathcal{G}_W(S_\ell \cup \{\mathbf{W}\})$ and player 1 wins $\mathcal{G}_W(T_i \cup \{\mathbf{W}\})$. Hence $T_i$ belongs to $\mathcal{A}$.

*Case 2*: $W \not\subseteq T_i$. Note that $T_i$ is also a 1-trap in the game $\mathcal{G}_W(S_\ell)$ and player 1 wins $\mathcal{G}_W(T_i)$. Hence $T_i$ belongs to $\mathcal{B}$.

Thus, $\mathcal{T} \subset \mathcal{S}$ and by hypothesis, player 1 wins all $\mathcal{G}(T_i)$.

If $S_\ell \in \mathcal{B}$ then player 1 wins $\mathcal{G}(S_\ell) = \mathcal{G}_W(S_\ell)$. Otherwise $S_\ell \in \mathcal{A}$ and player 1 wins $\mathcal{G}_W(S_\ell \cup \{\mathbf{W}\})$.

- If $S_\ell \notin \Omega$, then for all $v \in S_\ell$, $Attr_1(\{v\}, G(S_\ell)) = S_\ell$ or player 1 wins $\mathcal{G}(S_\ell \setminus Attr_1(\{v\}, G(S_\ell)))$ since $S_\ell \setminus Attr_1(\{v\}, G(S_\ell))$ is a 1-trap in the game $\mathcal{G}(S_\ell)$. By Lemma 11, player 1 wins $\mathcal{G}(S_\ell)$.
- Otherwise by Lemma 12, there is a 0-trap $Q \subset S_\ell \cup \{\mathbf{W}\}$ in $\mathcal{G}_W(S_\ell \cup \{\mathbf{W}\})$ such that player 1 wins $\mathcal{G}_W(Q)$.
  - If $\mathbf{W} \notin Q$ then $W \cap V_0 \cap Q = \emptyset$ and $Q$ also determines a 0-trap in the game $\mathcal{G}(S_\ell)$. Since $W \not\subseteq Q$, player 1 wins $\mathcal{G}(Q) = \mathcal{G}_W(Q)$ and let $Y = Q$.
  - If $\mathbf{W} \in Q$ then let $Y = Q \setminus \{\mathbf{W}\}$. Note that for all $v \in V_0 \cap W \cap Q$, $E_W(v) \cap Q = E_W(v) \cap (S_\ell \cup \{\mathbf{W}\})$ and $|E_W(v) \cap (S_\ell \cup \{\mathbf{W}\})| > 1$. Hence $Y$ determines a 0-trap in the game $\mathcal{G}(S_\ell)$. Since player 1 wins $\mathcal{G}_W(Y \cup \{\mathbf{W}\})$, $Y$ belongs to $\mathcal{A}$ and by hypothesis player 1 wins $\mathcal{G}(Y)$.

  Therefore there exists a 0-trap $Y$ in the game $\mathcal{G}(S_\ell)$ such that player 1 wins $\mathcal{G}(Y)$. Also $Attr_1(Y, G(S_\ell)) = S_\ell$ or player 1 wins $\mathcal{G}(S_\ell \setminus Attr_1(Y, G(S_\ell)))$ since $S_\ell \setminus Attr_1(Y, G(S_\ell))$ is a 1-trap in the game $\mathcal{G}(S_\ell)$. Then we construct a winning strategy for player 1 in the game $\mathcal{G}(S_\ell)$ as follows.
  - If the token is in $Y$ then player 1 forces the token in $Y$ forever and follows the winning strategy in $\mathcal{G}(Y)$.
  - If the token is in $Attr_1(Y, G(S_\ell))$ then player 1 forces the token to $Y$.
  - Otherwise, player 1 follows the winning strategy in $\mathcal{G}(S_\ell \setminus Attr_1(Y, G(S_\ell)))$.

By hypothesis, for all $S_i \in \mathcal{S}$, player 1 wins $\mathcal{G}(S_i)$. ◀

By Lemmas 15 and 16, we have the following theorem.

▶ **Theorem 17.** *$Win_0(\mathcal{G}) = Win_0(\mathcal{G}_W) \setminus \{\mathbf{W}\}$ and $Win_1(\mathcal{G}) = Win_1(\mathcal{G}_W) \setminus \{\mathbf{W}\}$.* ⌟

**Input**: An explicit Müller game $\mathcal{G} = (G, \Omega)$
**Output**: The winning regions of player 0 and player 1
topologically order $\Omega$;
$G' = (V_0', V_1', E') \leftarrow G = (V_0, V_1, E)$;
$\Omega' \leftarrow \Omega$;
$Win_0 \leftarrow \emptyset$;
**while** $\Omega' \neq \emptyset$ **do**
  $W_i' \leftarrow \text{pop}(\Omega')$
  **if** $\mathcal{G}'(W_i')$ is forced-connected **then**
    **if** $W_i'$ is a 1-trap in $\mathcal{G}'$ **then**
      remove $Attr_0(W_i', G')$ from $\mathcal{G}'$ and add it to $Win_0$;
    **else**
      $\mathcal{G}' \leftarrow \mathcal{G}'_{W_i'}$;
    **end**
  **end**
**end**
**return** $Win_0 \cap V$ and $V \setminus Win_0$

**Figure 1** Algorithm for explicit Müller games.

We briefly explain the algorithm, presented in Figure 1, that takes as input an explicit Müller game $\mathcal{G}$ and returns the winning regions of the players. Initially, the algorithm orders $\Omega$ topologically: $W_1 < W_2 < \ldots < W_s$. At each iteration, the algorithm modifies the arena and the winning conditions:

- If $W_i'$ doesn't determine a subgame in game $\mathcal{G}'$ or $\mathcal{G}'(W_i')$ isn't forced-connected, $W_i'$ is removed from $\Omega'$.
- Otherwise, $\mathcal{G}'(W_i')$ is forced-connected, then:
  - If $W_i'$ is a 1-trap in $\mathcal{G}'$ then $Attr_0(W_i', G')$ is removed from $\mathcal{G}'$ and added to the winning region of player 0. Note that all $W' \in \Omega'$ with $W' \cap Attr_0(W_i', G') \neq \emptyset$ are removed.
  - Otherwise, apply Horn's construction to $\mathcal{G}'$ by setting $\mathcal{G}' = \mathcal{G}'_{W_i'}$. In this construction, a new player 1's node $\mathbf{W}_i'$ is added to $G'$, $\mathbf{W}_i'$ is added to all supersets of $W_i'$ in $\Omega'$ and $W_i'$ itself is removed from $\Omega'$, which maintains the topological order of $\Omega'$.

Now our goal is to show that Horn's algorithm preserves the winner of the original games at each iteration so that the algorithm can compute the winning regions correctly. This is shown in the next lemma and theorem.

▶ **Lemma 18.** *At the end of each iteration, we have*

$$Win_0(\mathcal{G}) = (Win_0(\mathcal{G}') \cup Win_0) \cap V \text{ and } Win_1(\mathcal{G}) = Win_1(\mathcal{G}') \cap V.$$

**Proof.** Initially, $\mathcal{G}' = \mathcal{G}$ and $Win_0 = \emptyset$, which holds the lemma. Then for $i = 1, 2, \ldots, s$, we want to show that at the end of $i$th iteration, $Win_0(\mathcal{G}) = (Win_0(\mathcal{G}') \cup Win_0) \cap V$ and $Win_1(\mathcal{G}) = Win_1(\mathcal{G}') \cap V$. Let $\mathcal{G}''$ be $\mathcal{G}'$ and $Win_0'$ be $Win_0$ at the beginning of $i$th iteration. Let $\mathcal{G}'''$ be $\mathcal{G}'$ and $Win_0''$ be $Win_0$ at the end of $i$th iteration. By hypothesis, $Win_0(\mathcal{G}) = (Win_0(\mathcal{G}'') \cup Win_0') \cap V$ and $Win_1(\mathcal{G}) = Win_1(\mathcal{G}'') \cap V$. If $W_i'$ doesn't determine a subgame in game $\mathcal{G}'$ or $W_i'$ isn't forced-connected then by Lemma 13, $W_i'$ can be removed without affecting the winning regions of the players of the game. Otherwise, if $W_i'$ is a 1-trap in $\mathcal{G}'$ then player 0 wins $\mathcal{G}''(Attr_0(W_i', G''))$ by forcing the token to $W_i'$ and then to go through $W_i'$. Since $Attr_0(W_i', G'')$ is a 1-trap in $G''$, $Attr_0(W_i', G'') \subseteq Win_0(\mathcal{G}'')$. Since $\mathcal{G}''' = \mathcal{G}''(V'' \setminus Attr_0(W_i', G''))$, $Win_0(\mathcal{G}'') = Win_0(\mathcal{G}''') \cup Attr_0(W_i', G'')$ and $Win_1(\mathcal{G}'') = Win_1(\mathcal{G}''')$. Therefore, $Win_0(\mathcal{G}) = (Win_0(\mathcal{G}''') \cup Attr_0(W_i', G'') \cup Win_0') \cap V = (Win_0(\mathcal{G}''') \cup Win_0'') \cap V$ and $Win_1(\mathcal{G}) = Win_1(\mathcal{G}''') \cap V$. If $W_i'$ isn't a 1-trap in $\mathcal{G}'$ then by Theorem 17, $Win_0(\mathcal{G}'') = Win_0(\mathcal{G}''') \setminus \{\mathbf{W}_i'\}$ and $Win_1(\mathcal{G}'') = Win_1(\mathcal{G}''') \setminus \{\mathbf{W}_i'\}$. Therefore, $Win_0(\mathcal{G}) = (Win_0(\mathcal{G}''') \cup Win_0'') \cap V$ and $Win_1(\mathcal{G}) = Win_1(\mathcal{G}''') \cap V$. By hypothesis, at the end of each iteration, $\mathcal{G}'$ and $Win_0$ hold the lemma. ◀

By Lemma 18, we have the following theorem.

▶ **Theorem 19.** *At the end of the algorithm, we have*

$$Win_0(\mathcal{G}) = Win_0 \cap V \text{ and } Win_1(\mathcal{G}) = V \setminus Win_0.$$ ⌟

At each iteration, at most one player 1's vertex is added and at most $|V_0'|$ edges are added. Therefore, $|V_0'| = |V_0|$, $|V_1'|$ is bounded by $|V_1| + |\Omega|$ and $|E'|$ is bounded by $|E| + |V_0||\Omega|$. For time complexity of the algorithm, there are at most $|\Omega|$ iterations in a run and the most time-consuming operation is to determine if $\mathcal{G}'(W_i')$ is forced-connected. By Theorem 9 and Theorem 10, we have the following theorems.

▶ **Theorem 20.** *There exists an algorithm that solves the explicit Müller game $\mathcal{G}$ in time* $\mathbf{O}(|\Omega| \cdot ((\sqrt{|V_1| + |\Omega|} + 1)(|E| + |V_0||\Omega|) + (|V_1| + |\Omega|)^2))$. ⌟

▶ **Theorem 21.** *There exists an algorithm that solves the explicit Müller game $\mathcal{G}$ in time* $\mathbf{O}(|\Omega| \cdot (|V_0| + |V_1| + |\Omega|) \cdot |V_0| \log |V_0|)$. ⌟

Both of these algorithms beat the bound of Horn's algorithm. Importantly, Theorem 21 decreases the degree of $|\Omega|$ from $|\Omega|^3$ in Horn's algorithm to $|\Omega|^2$. Since $|\Omega|$ is bounded by $2^{|V|}$, the improvement is significant.

## 5 Applications

We now apply the results above to specific classes of games. Here we give three examples of such classes. The first such class is the class of fully separated Müller games. A Müller game $\mathcal{G}$ is *fully separated* if for each $W \in \Omega$ there is a $s_W$, called separator, such that all $s_W \in W$ but $s_W \notin W'$ for all $W' \in \Omega$ distinct from $W$. The second class of games is the class of linear games. A Müller game $\mathcal{G}$ is a *linear game* if the set $\Omega$ forms a linear order $W_1 \subset W_2 \subset \ldots \subset W_s$. These classes of games were studied in [15]. As the games are fully separated, when one constructs $\mathcal{G}'_{W'_i}$ there is no need to add a new vertex. Then applying Theorems 9 and 10 to Horn's algorithm, we get the following result:

▶ **Theorem 22.** *Each of the following is true:*
1. *Any fully separated Müller game $\mathcal{G}$ can be solved in time* $\mathbf{O}(|V| \cdot ((\sqrt{|V_1|}+1)|E| + |V_1|^2))$.
2. *Any fully separated Müller game $\mathcal{G}$ can be solved in time* $\mathbf{O}(|V|^2 \cdot |V_0| \log |V_0|)$. ⌟

Both of these algorithms beat the bound of [15] $\mathbf{O}(|V|^2|E|)$ that solves fully separated Müller game. Applying Theorem 20 and Theorem 21, we have the following theorems.

▶ **Theorem 23.** *Each of the following is true:*
1. *Any linear Müller game $\mathcal{G}$ can be solved in time* $\mathbf{O}(|V| \cdot ((\sqrt{|V|}+1) \cdot |V_0||V| + |V|^2))$.
2. *Any linear Müller game $\mathcal{G}$ can be solved in time* $\mathbf{O}(|V|^2 \cdot |V_0| \log |V_0|)$. ⌟

Both of these algorithms beat the bound $\mathbf{O}(|V|^{2\cdot|V|-1}|E|)$ from of [15] and the bound $\mathbf{O}(|V|^3 \cdot |V_0|)$ implied from Horn's algorithm.

The third class of Müller games was introduced by A. Dawar and P. Hunter in [14]. They investigated games with anti-chain winning condition. A winning condition $\Omega$ is an *anti-chain* if $X \not\subseteq Y$ for all $X, Y \in \Omega$. Applying Theorem 20 and Theorem 21, we have the following theorems. Note that, since the winning condition is an anti-chain, $|V'_1|$ is bounded by $|V_1|$, $|E'|$ is bounded by $|E|$ and no new player 1's vertex is added to $\Omega'$.

▶ **Theorem 24.** *Each of the following is true:*
1. *Any Müller game $\mathcal{G}$ with anti-chain winning condition can be solved in time* $\mathbf{O}(|\Omega| \cdot ((\sqrt{|V_1|}+1)|E| + |V_1|^2))$.
2. *Any Müller game $\mathcal{G}$ with anti-chain winning condition can be solved in time* $\mathbf{O}(|\Omega||V| \cdot |V_0| \log |V_0|)$. ⌟

Just as above, both of the algorithms beat the bound $\mathbf{O}(|\Omega||V|^2|E|)$ from [14] and the bound of Horn's algorithm $\mathbf{O}(|\Omega||V||E|)$ that solves the explicit Müller games with anti-chain winning conditions.

## 6 A note on Horn's proof

In [12], Horn considers sensible sets. A winning set $W \in \Omega$ is *sensible* if it determines a subgame. Initially, all non-sensible sets are removed (this is fine). Then Horn's assumption is that through the iteration process (in the algorithm in figure 1) sensibility is preserved.

When $\mathcal{G}_W$ is built (during iterations), a winning condition $W'$ that contains $W$ might become non-sensible. Hence, sensibility is not preserved during iterations. Horn's analysis doesn't take non-sensible sets into account. This is important. In Horn's defence, assume we remove all non-sensible winning sets in the current game $\mathcal{G}_W$ (Note that the algorithm removes non-sensible winning conditions). However, Horn does not prove that this is a correct action. Horn's proof does not analyse an intricate interplay between sensibility and maintenance of the winning sets at each iteration. Neglecting non-sensible sets makes the proofs of Lemmas 6 and 7 (in [12]) incorrect. Here is an example.

Let $\mathcal{G} = (G, \Omega)$ be a game where $G$ is shown in figure 2 and $\Omega = \{W_1, W_2, W_3\}$, where $W_1 = \{v_1, u_1\}$, $W_2 = \{v_1, u_1, u_2\}$, and $W_3 = \{v_1, v_2, u_1, u_2\}$. During the algorithm, player 0 wins the subgame determined by $W_1$. The set $W_1$ isn't a 1-trap. So the new player 1's node $\mathbf{W}_1$ is added. The sensible set $W_2$ now becomes non-sensible in $\mathcal{G}_{W_1}$. Let us assume that $W_2 \cup \{\mathbf{W}_1\}$ is removed from the winning set in $\mathcal{G}_{W_1}$. Then player 0 wins the subgame determined by $W_3 \cup \{\mathbf{W}_1\}$. Horn's Lemma 7 applied to the game in $\mathcal{G}_{W_1}$ that occurs on $W_3 \cup \{\mathbf{W}_1\}$ states the following.

*Player 1 wins the original game played on $W_3$, where $W_3$ is removed from $\Omega$.*

The proof uses induction that has the following important (in our view unrecoverable) flaw. The proof considers the maximal winning sets inside $W_3$. Clearly, the maximal winning set inside $W_3$ is $W_3$ itself. Horn refers to player 1 winning strategy on $W_3$. Such strategy does not exists as it needs to be built. This is a self-loop argument. Trying to save Horn's proof, lets assume that $W_2$ was considered by Horn to be the maximal set. Since $W_2 \cup \{\mathbf{W}_1\}$ is non-sensible in $\mathcal{G}_{W_1}$, it is removed during the algorithm. Horn claims that player 1 has a winning strategy by playing inside $W_2$ and uses it to build a winning strategy in $W_3$. However, player 0 wins $\mathcal{G}(W_1)$ and $W_1$ is a 1-trap in $\mathcal{G}(W_2)$. As a result, player 1 has no winning strategy in $\mathcal{G}(W_2)$ and Horn's proof fails. Since Horn reuses the proof of Lemma 7 in the proof of Lemma 6, Horn fails on the proofs of Lemmas 6 and 7. These show that Horn's inductive arguments fail.

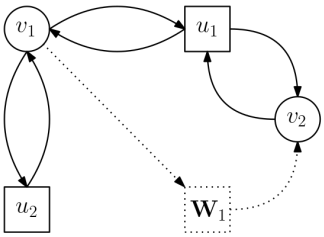



**Figure 2** The counter case to Horn's Lemmas 6 and 7.

Our section 4 develops new ideas and methods that are not present in Horn's arguments. As an example, we consider extendible and non-extendible sets in the proof of Lemma 16. The lemma takes winning regions that are subsets of non-sensible sets into account. These are placed in set $\mathcal{B}$ in the lemma. We also show that the players' winning nodes stay invariant with each iteration. This completely differs from Horn's arguments. Horn's Lemmas 5, 6, and 7 are aimed at proving that the original game restricted to $W$, given by the iteration, is won by one of the players. Our approach is obviously different.

## References


1 Michael A Bender, Jeremy T Fineman, and Seth Gilbert. A new approach to incremental topological ordering. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1108–1115. SIAM, 2009.

2 Michael A Bender, Jeremy T Fineman, Seth Gilbert, and Robert E Tarjan. A new approach to incremental cycle detection and related problems. *arXiv preprint*, 2011. `arXiv:1112.0784`.

3 Hans L Bodlaender, Michael J Dinneen, and Bakhadyr Khoussainov. On game-theoretic models of networks. In *Algorithms and Computation: 12th International Symposium, ISAAC 2001 Christchurch, New Zealand, December 19–21, 2001 Proceedings 12*, pages 550–561. Springer, 2001.

4 Hans L Bodlaender, Michael J Dinneen, and Bakhadyr Khoussainov. Relaxed update and partition network games. *Fundamenta Informaticae*, 49(4):301–312, 2002.

5 Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 252–263, 2017.

6 Krishnendu Chatterjee and Monika Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *Journal of the ACM (JACM)*, 61(3):1–40, 2014.

7 Michael J Dinneen and Bakhadyr Khoussainov. Update networks and their routing strategies. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 127–136. Springer, 2000.

8 Stefan Dziembowski, Marcin Jurdzinski, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 99–110. IEEE, 1997.

9 Aniruddh Gandhi, Bakhadyr Khoussainov, and Jiamou Liu. Efficient algorithms for games played on trees with back-edges. *Fundamenta Informaticae*, 111(4):391–412, 2011.

10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke. Automata, logics, and infinite games. lncs, vol. 2500, 2002.

11 Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert E Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. *ACM Transactions on Algorithms (TALG)*, 8(1):1–33, 2012.

12 Florian Horn. Explicit muller games are ptime. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

13 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *International Symposium on Mathematical Foundations of Computer Science*, pages 495–506. Springer, 2005.

14 Paul Hunter and Anuj Dawar. Complexity bounds for muller games. *Theoretical Computer Science (TCS)*, 2008.

15 Hajime Ishihara and Bakhadyr Khoussainov. Complexity of some infinite games played on finite graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 270–281. Springer, 2002.

16 Imran Khaliq, Bakhadyr Khoussainov, and Jiamou Liu. Extracting winning strategies in update games. In *Models of Computation in Context: 7th Conference on Computability in Europe, CiE 2011, Sofia, Bulgaria, June 27-July 2, 2011. Proceedings 7*, pages 142–151. Springer, 2011.

17 Bakhadyr Khoussainov, Jiamou Liu, and Imran Khaliq. A dynamic algorithm for reachability games played on trees. In *Mathematical Foundations of Computer Science 2009: 34th International Symposium, MFCS 2009, Novy Smokovec, High Tatras, Slovakia, August 24-28, 2009. Proceedings 34*, pages 477–488. Springer, 2009.

18 Donald A Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

19 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

**20** Anil Nerode, Jeffrey B Remmel, and Alexander Yakhnis. Mcnaughton games and extracting strategies for concurrent programs. *Annals of Pure and Applied Logic*, 78(1-3):203–242, 1996.

**21** Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

**22** Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

# On the Perturbation Function of Ranking and Balance for Weighted Online Bipartite Matching

**Jingxun Liang**
IIIS, Tsinghua University, Beijing, China

**Zhihao Gavin Tang**
ITCS, Shanghai University of Finance and Economics, China

**Yixuan Even Xu**
IIIS, Tsinghua University, Beijing, China

**Yuhao Zhang**
Shanghai Jiao Tong University, China

**Renfei Zhou**
IIIS, Tsinghua University, Beijing, China

## Abstract

Ranking and Balance are arguably the two most important algorithms in the online matching literature. They achieve the same optimal competitive ratio of $1 - 1/e$ for the integral version and fractional version of online bipartite matching by Karp, Vazirani, and Vazirani (STOC 1990) respectively. The two algorithms have been generalized to weighted online bipartite matching problems, including vertex-weighted online bipartite matching and AdWords, by utilizing a perturbation function. The canonical choice of the perturbation function is $f(x) = 1 - e^{x-1}$ as it leads to the optimal competitive ratio of $1 - 1/e$ in both settings.

We advance the understanding of the weighted generalizations of Ranking and Balance in this paper, with a focus on studying the effect of different perturbation functions. First, we prove that the canonical perturbation function is the *unique* optimal perturbation function for vertex-weighted online bipartite matching. In stark contrast, all perturbation functions achieve the optimal competitive ratio of $1 - 1/e$ in the unweighted setting. Second, we prove that the generalization of Ranking to AdWords with unknown budgets using the canonical perturbation function is at most 0.624 competitive, refuting a conjecture of Vazirani (2021). More generally, as an application of the first result, we prove that no perturbation function leads to the prominent competitive ratio of $1 - 1/e$ by establishing an upper bound of $1 - 1/e - 0.0003$. Finally, we propose the online budget-additive welfare maximization problem that is intermediate between AdWords and AdWords with unknown budgets, and we design an optimal $1 - 1/e$ competitive algorithm by generalizing Balance.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online Matching, AdWords, Ranking, Water-Filling

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.80

**Related Version** *Full Version*: https://arxiv.org/abs/2210.10370

**Supplementary Material**
*Software (Source Code)*: https://github.com/orbitingflea/perturbation-function
archived at swh:1:dir:24101e3982129e07e5c1f644f0ff611800a98730

**Funding** This work is supported by Science and Technology Innovation 2030 – "New Generation of Artificial Intelligence" Major Project No.(2018AAA0100903), Program for Innovative Research Team of Shanghai University of Finance and Economics (IRTSHUFE) and the Fundamental Research Funds for the Central Universities.
*Zhihao Gavin Tang*: Zhihao Gavin Tang is partially supported by Huawei Theory Lab.
*Yuhao Zhang*: Yuhao Zhang is supported by National Natural Science Foundation of China, Grant No. 62102251.

**Acknowledgements** We thank all anonymous reviewers for their insightful comments.

## 1 Introduction

Online bipartite matching has been extensively studied since the seminal work of Karp, Vazirani, and Vazirani [22]. Two remarkable algorithms, Ranking of Karp et al. [22] and Balance of Kalyanasundaram and Pruhs [19], achieve the same optimal competitive ratio of $1 - 1/e$ for the integral (randomized) and fractional (deterministic) version of the problem respectively.

**AdWords and Vertex-weighted.** Motivated by the display of advertising on the Internet, Mehta et al. [25] generalized the online bipartite matching problem so that it allows weighted graphs. Consider an underlying bipartite graph $G = (L \cup R, E)$ with $L, R$ being offline and online vertices. Each vertex $u \in L$ is associated with a budget $B_u$, and each edge $(u, v) \in E$ is associated with a bid $w_{uv}$. The offline vertices and their corresponding budgets are known in advance. The online vertices arrive one at a time, with their incident edges and associated bids, and have to be matched immediately and irrevocably to some $u \in L$. An offline vertex $u \in L$ can be matched to multiple online vertices. Let $S_u$ be the set of online vertices matched to $u$ and then, the revenue of $u$ equals $\min\{B_u, \sum_{v \in S_u} w_{uv}\}$, that is, the revenue cannot exceed the budget. The goal is to maximize the total revenue and to compete against the optimal revenue of an offline algorithm that knows the whole graph.

Mehta et al. [25] established an optimal $(1 - 1/e)$-competitive algorithm for the fractional version[1] of the problem by generalizing Balance to Perturbed-Balance. Later, Aggarwal et al. [1] considered a restricted setting of AdWords, called vertex-weighted online bipartite matching, in which all edges incident to $u$ have the same weight of $w_u = B_u$. They generalized Ranking to Perturbed-Ranking and obtained the same $1 - 1/e$ competitive ratio for the integral version of the problem.

The two generalizations are both greedy-based algorithms with a careful perturbation of the weights. Specifically, upon the arrival of each vertex $v$, the algorithm matches it with the offline vertex $u$ with the maximum perturbed weight $f(x_u) \cdot w_{uv}$, among those neighbors whose budgets have not yet been exhausted. Here, $x_u$ corresponds to the random rank of $u$ in Perturbed-Ranking and the fraction of budget spent so far in Perturbed-Balance. The canonical choice of the perturbation function is $f(x) = 1 - e^{x-1}$, applied by Mehta et al. [25] and Aggarwal et al. [1]. Notably, Devanur, Jain, and Kleinberg [7] provided a unified primal-dual analysis for Perturbed-Ranking and Perturbed-Balance, in which the perturbation function plays a critical role even for the unweighted online bipartite matching problem.

Despite the above successful generalizations of Ranking and Balance from unweighted to weighted graphs, we lack an understanding of the extra difficulty introduced by weighted graphs. In this paper, we revisit the two classic algorithms and focus on the perturbation function. Notice that for unweighted graphs, Perturbed-Ranking (resp. Perturbed-Balance) with an arbitrary perturbation function degenerates to the same Ranking (resp. Balance) algorithm and achieves the optimal competitive ratio of $1 - 1/e$. We examine the importance of perturbation functions by studying the performance of Perturbed-Ranking and Perturbed-Balance on weighted graphs with an arbitrary perturbation function.

Our first result confirms the importance of the perturbation function, proving that the canonical perturbation function $f(x) = 1 - e^{x-1}$ is the unique optimal perturbation function for vertex-weighted online bipartite matching.

[1] The paper states their result with the small-bid assumption, i.e., $\gamma = \max_{u,v} w_{uv}/B_u$ is small. We consider the fractional AdWords problem, corresponding to the case when $\gamma \to 0$. See our discussion in Section 2.

▶ **Theorem 1.** *The perturbation function* $f(x) = 1 - e^{x-1}$ *is unique (up to a scale factor) for* Perturbed-Ranking *and* Perturbed-Balance *to achieve the optimal competitive ratio of* $1 - 1/e$ *for vertex-weighted online bipartite matching.*

It is surprising that this question has been overlooked by the online matching community. We introduce a new family of hard instances that heavily exploits the power of weighted graphs. Noticeably, prior to our work, the only $1 - 1/e$ impossibility result by Karp et al. [22] is established for unweighted graphs. This advanced understanding is also the starting point for us to explore the limitation of Perturbed-Ranking in a more general model, i.e., AdWords with unknown budgets.

**AdWords with Unknown Budgets.** A major open question left by Mehta et al. [25] is the competitive ratio of Perturbed-Ranking for the fractional AdWords problem. In addition to obvious theoretical interests, the Perturbed-Ranking algorithm has a merit of budget-obliviousness, as pointed out by Vazirani [28] and Udwani [27]. I.e., the algorithm does not need to know the budget of each vertex, in contrast to the Perturbed-Balance algorithm. Formally, consider the setting of AdWords with unknown budgets: the algorithm has no prior knowledge of the budgets and only learns the budget of each vertex $u$ when the total revenue of $u$ first exceeds its budget. Observe that Perturbed-Balance is not applicable in this setting, since its decision at each step depends on the fraction of budget spent on each offline vertex.

Perturbed-Ranking is the only known algorithm for AdWords with unknown budgets so far. Using the canonical perturbation function $f(x) = 1 - e^{x-1}$, Vazirani [28] proved Perturbed-Ranking is $(1 - 1/e)$-competitive assuming a no-surpassing property. Udwani [27] proved that the algorithm is 0.508-competitive in the general case and is 0.522-competitive with a different perturbation function $f(x) = 1 - e^{1.15(x-1)}$.

It is natural to ask if other perturbation functions can lead to a better competitive ratio, or even $1 - 1/e$. In this paper, we give a limitation of *all* perturbation functions, showing a separation between vertex-weighted online bipartite matching and AdWords on the performance of Perturbed-Ranking.

We first show that Perturbed-Ranking with the canonical perturbation function can only achieve a competitive ratio of at most $0.624 < 1 - 1/e$. Then, together with the family of instances we constructed in the proof of Theorem 1, we manage to prove that any perturbation function cannot lead to the prominent competitive ratio of $1 - 1/e$:

▶ **Theorem 2.** *The competitive ratio of* Perturbed-Ranking *algorithm with any perturbation function* $f(x)$ *on fractional AdWords is at most* $1 - 1/e - 0.0003$. *In particular, using the canonical function* $f(x) = 1 - e^{x-1}$, *the competitive ratio is at most* 0.624.

Our result refutes the conjecture of Vazirani [28] that Perturbed-Ranking is $1 - 1/e$ competitive. Moreover, our construction is clean and simple, suggesting that the no-surpassing assumption might be too strong to hold in reality. Such result leads to the conjecture that there is no $1 - 1/e$ competitive algorithm for AdWords with unknown budgets.

**Remark.** Very recently, independently by our work, Udwani [27] updated his paper and proved that a specific perturbation function family $f(x) = 1 - e^{\beta(x-1)}$ is at most 0.624-competitive for any $\beta > 0$. Our result provides a stronger observation by another approach that shows *all* perturbation functions cannot achieve the competitive ratio of $1 - 1/e$.

**Online Budget-additive Welfare Maximization.** The above upper bound of the competitive ratio for Perturbed-Ranking suggests that AdWords with unknown budgets should be strictly harder than AdWords, in terms of the worse-case competitive ratio. Unfortunately, our current construction is specific to the Perturbed-Ranking algorithm and does not serve as a problem hardness.

Inspired by the online submodular welfare maximization problem (that we discuss below in the related work section), we consider a variant which we call online budget-additive welfare maximization problem, that lies in between the original AdWords and AdWords with unknown budgets. Specifically, we assume that 1) the algorithm has no information of the budgets at the beginning, 2) at each step, the algorithm can query for each vertex $u$, the value of $w_u(S) = \min\{B_u, \sum_{v \in S} w_{uv}\}$ for any subset $S$ of arrived vertices. Notice that AdWords with unknown budgets can be interpreted in a similar way, except that the algorithm can only query those sets $S$ that are subsets of $S(u) \cup \{v\}$ where $S(u)$ is the set of current matched vertices to $u$.

Our final result is an optimal algorithm for the fractional version of the problem. We hope it would shed some light on designing online algorithms beyond Perturbed-Ranking in the AdWords with unknown budgets setting and designing algorithms beyond greedy (with unrestricted computational power) in the online submodular welfare maximization setting.

▶ **Theorem 3.** *There exists a fractional algorithm that achieves the competitive ratio of* $(1 - 1/e)$ *for the Online Budget-Additive Welfare Maximization problem.*

**Roadmap.** Section 2 presents the formal definitions of the Perturbed-Ranking and Perturbed-Balance algorithms. We prove Theorem 1 in Section 3 and Theorem 2 in Section 4. Due to space limit, we provide the proof of Theorem 3 in the full version.

## 1.1 Related Work

The seminal work of Karp et al. [22] studied the unweighted and one-sided online bipartite matching model and proposed the optimal $(1 - 1/e)$-competitive algorithm: Ranking. The analysis of Ranking has been refined and simplified by a series of works [2, 7, 11, 8]. Kalyanasundaram and Pruhs studied the $b$-matching model and designed Balance (fractional) that also achieves the competitive ratio of $1 - 1/e$. The model has been generalized to many weighted variants, e.g., vertex-weighted [1, 14, 15, 18], edge-weighted [3, 9, 10], and AdWords [6, 25, 17]. Besides the aforementioned generalization of Ranking and Balance to the vertex-weighted and AdWords settings, Huang et al. [15] generalized Ranking to the vertex-weighted setting with random arrival order, by utilizing a two-dimensional perturbation function. They achieved a competitive ratio of 0.653 that is subsequently improved to 0.662 by Jin and Williamson [18]. Another line of work adapts Ranking and Balance to other matching problems, including online bipartite matching with random arrivals [21, 24], oblivious matching [5, 26] and fully online matching [12, 13, 16].

The most general extension of online bipartite matching is the online submodular welfare maximization problem. It captures most of the weighted variants of online bipartite matching discussed above. In this problem, a set of $n$ offline vertices are given, each associated with a

monotone submodular function $w_u$. Upon the arrival of an online vertex, it must be assigned to one of the offline vertices and the goal is to maximized the welfare $\sum_u w_u(S_u)$, where $S_u$ is the set of vertices received by $u$. The algorithm is assumed to have value oracles for the functions. I.e., an algorithm can query the value of $w_u(S)$ for an arbitrary subset $S$ of arrived online vertices. Kapralov et al. [20] proved that the 0.5-competitive greedy algorithm is optimal with restricted computational powers. For the unknown i.i.d. setting, they provided an optimal $(1-1/e)$-competitive algorithm. In the random arrival model, Korula et al. [23] proved that greedy is at least 0.5052-competitive, and the ratio is improved to 0.5096 by Buchbinder et al. [4]. Our budget-additive welfare maximization problem is a special case of the submodular welfare maximization problem where every $w_u$ is a budget-additive function and admits an $(1-1/e)$-competitive algorithm.

Moreover, the AdWords with unknown budgets problem suggests us to study a more restricted oracle access for online submodular welfare maximization. We call it *marginal oracle*, that on the arrival of an online vertex $v$, the algorithm can only query the value of $w_u(S)$ for $S \subseteq S_u(v) \cup \{v\}$, where $S_u(v)$ is the current matched vertex set to $u$. Based on our discoveries in this paper, we make the following three conjectures for future work:

- Online submodular welfare maximization with marginal oracles does *not* admit a $1-1/e$ competitive algorithm.
- AdWords with unknown budgets does *not* admit a $1-1/e$ competitive algorithm.
- Online submodular welfare maximization admits a $0.5+\Omega(1)$ competitive algorithm.

All the three conjectures assume unlimited computational powers so that the third conjecture does not violate the impossibility result of [20]. Observe that if the second conjecture holds, it automatically confirms the first conjecture, and implies a price of budget-obliviousness for AdWords.

## 2 Preliminaries

We first give the formal definitions of Perturbed-Ranking and Perturbed-Balance for the vertex-weighted online bipartite matching problem and then discuss their extensions to the fractional AdWords problem. Both algorithms depends on a perturbation function.

▶ **Definition 4** (Perturbation Function). *A perturbation function is a non-increasing and right continuous function* $f(x) : [0,1] \to [0,1]$.

### 2.1 Vertex-weighted

Given a perturbation function $f$, the two algorithms are defined as below. Observe that Perturbed-Ranking is a randomized algorithm and Perturbed-Balance is a deterministic algorithm.

▶ **Definition 5** (Perturbed-Ranking [1]). *Sample a rank $y_u$ for each offline vertex $u \in L$ independently from a uniform distribution on $[0,1]$. On the arrival of an online vertex $v$, we match $v$ to the unmatched neighbor $u$ with maximum perturbed weight $f(y_u) \cdot w_u$.*

▶ **Definition 6** (Perturbed-Balance). *On the arrival time of an online vertex $v$, we continuously match $v$ to the offline neighbor $u$ with maximum perturbed weight $f(x_u) \cdot w_u$, where $x_u$ is the current matched portion of $u$.*

We remark that in the context of Perturbed-Ranking, a perturbation function can be interpreted as an alternative representation of a $[0,1]$-bounded random variable, in which $f(x)$ corresponds to the value of a quantile $x$. Moreover, the right continuity is necessary for the Perturbed-Balance algorithm to be well-defined.

## 2.2 AdWords

In Section 4, we shall work on the fractional version of AdWords (and its variant) that is (slightly) more relaxed than the AdWords problem with small bid assumption. See e.g. Udwani [27] for a more detailed discussion.

**Fractional AdWords.** The fractional AdWords problem allows each edge $(u, v)$ to be fractionally matched by an amount of $x_{uv} \in [0, 1]$, as long as the total matched portion of each online vertex $v$ is no more than a unit, i.e., $\sum_u x_{uv} \leq 1$.

Consider the following generalizations of Perturbed-Ranking and Perturbed-Balance for the fractional AdWords problem:

▶ **Definition 7** (Perturbed-Ranking [28, 27]). *Sample a rank $y_u$ for each offline vertex $u \in L$ independently from a uniform distribution on $[0, 1]$. On the arrival of an online vertex $v$, we* continuously *match $v$ to the neighbor $u$ with maximum perturbed weight $f(y_u) \cdot w_{uv}$, among those neighbors whose budgets have not been exhausted yet.*

▶ **Definition 8** (Perturbed-Balance (a.k.a. MSVV [25])). *On the arrival time of an online vertex $v$, we continuously match $v$ to the offline neighbor $u$ with maximum perturbed weight $f(x_u/B_u) \cdot w_{uv}$, where $x_u$ is the current used budget of $u$.*

# 3 Vertex-Weighted

In this section, we consider vertex-weighted online bipartite matching. We prove that to achieve the optimal competitive ratio of $\Gamma = 1 - 1/e$, the canonical choice of the perturbation function $f(x) = 1 - e^{x-1}$ is unique (up to a scale factor).

Our result holds for both Perturbed-Ranking and Perturbed-Balance. Indeed, we establish a dominance of Perturbed-Balance over Perturbed-Ranking in terms of worst-case competitive ratio.

▶ **Lemma 9.** *For any perturbation function $f$, the competitive ratio of* Perturbed-Ranking *is at most the competitive ratio of* Perturbed-Balance *for vertex-weighted online bipartite matching.*

We sketch our proof below and provide the detailed proof in the full version.

**Proof Sketch.** Given an arbitrary instance $G = (L \cup R, E, w)$, we construct an instance $G'$ so that the competitive ratio of Perturbed-Balance for $G$ and the competitive ratio of Perturbed-Ranking for $G'$ are approximately the same.

- For each offline vertex $u \in L$, create $N$ duplicates $\{u^{(i)}\}_{i=1}^N$ in $G'$ and with weights $w_{u^{(i)}} = w_u$.
- For each online vertex $v \in R$, create $N$ duplicates $\{v^{(i)}\}_{i=1}^N$ in $G'$ that arrive in a sequence.
- For each $(u, v) \in E$, let there be a complete bipartite graph between $\{u^{(i)}\}$ and $\{v^{(i)}\}$ in $G'$.

Now, we consider the behavior of Perturbed-Ranking on $G'$. Intuitively, although the ranks are drawn independently for each offline vertex, the set of $N$ ranks of $\{u^{(i)}\}_{i=1}^N$ should be "close" to $\left\{\frac{1}{N}, \frac{2}{N}, \ldots, 1\right\}$ with high probability when $N$ is sufficiently large. Formally, we prove the following mathematical fact in the full version.

▶ **Lemma 10.** *Let $x_1, \ldots, x_n$ be i.i.d. random variables sampled from $[0, 1]$ uniformly. Let $y_i$ be the $i$-th order statistics of $\{x_1, \ldots, x_n\}$, for $i = 1, \ldots, n$. Then for any parameter $\varepsilon$ with $4n^{-1/4} < \varepsilon < 1$, we have*

$$\Pr_{x_1,\ldots,x_n}\left[\left|y_i - \frac{i}{n}\right| \leq \varepsilon,\ \forall i \in [n]\right] \geq 1 - 2ne^{-\sqrt{n}/6}. \tag{3.1}$$

For now, we assume the set of ranks of $\{u^{(i)}\}$ is $\{\frac{1}{N}, \frac{2}{N}, \ldots, 1\}$ for each vertex $u$. Then, upon the arrival of $\{v^{(i)}\}$, we are basically running a discretized version of the Perturbed-Balance algorithm, with a step size of $\frac{1}{N}$. To formalize this intuition, we can introduce a family of $\varepsilon$-approximate Perturbed-Balance algorithms that approaches the behavior of Perturbed-Balance when $\varepsilon \to 0$. Moreover, we prove that the Perturbed-Ranking algorithm can be interpreted as an $\varepsilon$-approximate Perturbed-Balance algorithm when the ranks of $\{u^{(i)}\}$ behave nicely (which is of high probability). Finally, we conclude the proof of the lemma by letting $N$ go to infinity. The detail is provided in the full version. ◀

Equipped with the above lemma, we hereafter focus on the easy-to-analyze Perturbed-Balance algorithm, since it is deterministic while Perturbed-Ranking is randomized.

Our main construction is a family of instances that strongly restricts the shape of the perturbation function. Naturally, our construction is built upon the classical upper triangle graph that gives the tight $1 - 1/e$ competitive ratio for unweighted online bipartite matching. On the other hand, our construction consists of a few novel gadgets that might be useful for other weighted online matching problems. The following lemma also serves as a stepping stone of our result for the AdWords with unknown budget problem in Section 4.

▶ **Lemma 11.** *If the* Perturbed-Balance *algorithm achieves a competitive ratio of* $\Gamma$ *for vertex-weighted online bipartite matching, then the perturbation function* $f$ *satisfies the following:*

$$\left(\beta + 1 - e^{\beta-1} - \Gamma\right) \cdot f(\alpha) \geq (1 - (1-\Gamma) \cdot e^{\alpha}) \cdot \int_0^{\beta} f(x)\mathrm{d}x, \quad \forall \alpha, \beta \in [0,1]. \tag{3.2}$$

We defer its proof till the end of the section and proceed by first proving our main theorem.

▶ **Theorem 1.** *The perturbation function* $f(x) = 1 - e^{x-1}$ *is unique (up to a scale factor) for* Perturbed-Ranking *and* Perturbed-Balance *to achieve the optimal competitive ratio of* $1 - 1/e$ *for vertex-weighted online bipartite matching.*

**Proof.** By Lemma 9, it suffices to prove the theorem for Perturbed-Balance. By Lemma 11 with $\Gamma = 1 - 1/e$, the perturbation function $f(x)$ satisfies the following:

$$\frac{f(\alpha)}{1 - e^{\alpha-1}} \geq \frac{\int_0^{\beta} f(x)\mathrm{d}x}{\beta - e^{\beta-1} + e^{-1}}, \quad \forall \alpha, \beta \in (0,1).$$

Let $M \overset{\text{def}}{=\!=} \inf_{\alpha \in (0,1)} \frac{f(\alpha)}{1-e^{\alpha-1}}$. We must then have

$$f(\alpha) \geq M(1 - e^{\alpha-1}), \quad \forall \alpha \in [0,1], \tag{3.3}$$

$$\int_0^{\beta} f(x)\mathrm{d}x \leq M(\beta - e^{\beta-1} + e^{-1}), \quad \forall \beta \in [0,1]. \tag{3.4}$$

Taking the integral of $f(x)$ and applying (3.3), we have

$$\int_0^{\beta} f(x)\mathrm{d}x \geq M \int_0^{\beta} (1 - e^{x-1})\mathrm{d}x = M(\beta - e^{\beta-1} + e^{-1}).$$

Together with (3.4), we conclude that $f(x) = M(1 - e^{x-1})$ for all $x \in [0,1]$ according to the right-continuity of function $f$. Specifically, if there exists an $x^* \in [0,1)$, that $f(x^*) = M(1 - e^{x^*-1}) + \varepsilon$ for some $\varepsilon > 0$, then there exists a sufficiently small $\delta > 0$ such that for any $x \in [x^*, x^* + \delta]$, $f(x) \geq M(1 - e^{x-1}) + \frac{\varepsilon}{2}$. Then we can see by (3.3) that

$$\int_0^1 f(x)\mathrm{d}x \geq \int_0^1 \Big(M(1-e^{x-1}) + \mathbb{1}[x^* \leq x \leq x^* + \delta] \cdot \frac{\varepsilon}{2}\Big)\mathrm{d}x = \frac{M}{e} + \frac{\varepsilon\delta}{2},$$

which violates the statement of (3.4), $\int_0^1 f(x)\mathrm{d}x \leq \frac{M}{e}$. Therefore, $\forall x \in [0,1)$, $f(x) = M(1-e^{x-1})$. Also, for $f(1)$, note that $f(x)$ is decreasing, so $f(1) \leq \lim_{x\to 1^-} f(x) = 0$. Then $f(1) = 0$.

This shows that $\forall x \in [0,1]$, $f(x) = M(1-e^{x-1})$, concluding the proof of the theorem. ◀

## 3.1 Proof of Lemma 11

Fix $\alpha, \beta \in [0,1]$. Let $n, m$ be sufficiently large numbers. Refer to Figure 3.1 for our instance.

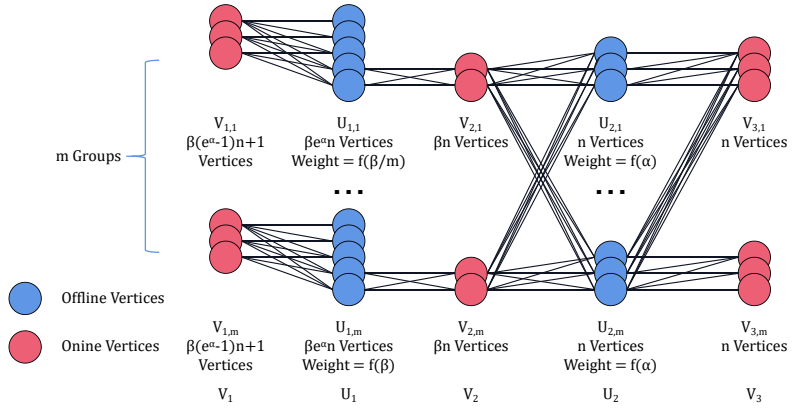



**Figure 3.1** Instance 1.

Our construction consists of $m$ groups of vertices, and each group consists of 5 parts. We use $V_{1,i}, V_{2,i}, V_{3,i}$ to denote the three online parts of group $i$ and $U_{1,i}, U_{2,i}$ to denote the two offline parts of group $i$. Let $V_j = \cup_{i\in m} V_{j,i}$ for $j \in \{1,2,3\}$ and $U_j = \cup_{i\in m} U_{j,i}$ for $j \in \{1,2\}$. We first define the vertices of the graph: for each $i \leq m$,

- $U_{1,i}$ consists of $(\beta e^\alpha n + 1)$ offline vertices[2] with the same weight of $f(\frac{i}{m} \cdot \beta)$.
- $U_{2,i}$ consists of $n$ offline vertices with the same weight of $f(\alpha)$.
- $V_{1,i}, V_{2,i}, V_{3,i}$ consist of $\beta(e^\alpha - 1)n, \beta n, n$ online vertices, respectively.

The arrival order of the vertices is the following:

$$V_{1,1} \to V_{1,2} \to \cdots \to V_{1,m} \to V_{2,1} \to V_{2,2} \to \cdots \to V_{2,m} \to V_{3,1} \to V_{3,2} \to \cdots \to V_{3,m}.$$

Next, we define the edges of the graph:

- $V_{1,i}$ and $U_{1,i}$ are connected as an *upper triangle*. I.e., the $j$-th vertex of $V_{1,i}$ is connected to the $k$-th vertex of $U_{1,i}$ if and only in $k \geq j$.
- $V_{2,i}$ and the last $\beta n$ vertices in $U_{1,i}$ are *fully connected*.
- $V_2$ and $U_2$ are *fully connected*.

[2] When $\beta e^\alpha n$ is a fraction, let there be $\lceil \beta e^\alpha n \rceil$ vertices. Since we are interested in the case when $n, m$ are sufficiently large, we safely omit the ceiling function for the simplicity of notation. We apply a similar treatment for fractions throughout the paper.

- $V_3$ and $U_2$ are connected as a *upper triangle*. I.e., the $j$-th vertex of $V_3$ is connected to the $k$-th vertex of $U_2$ if and only in $k \geq j$.

We first calculate the optimum matching of the graph. That is, matching together $(V_3, U_2)$ and $(V_1 \cup V_2, U_1)$. Therefore, we have

$$\mathrm{OPT} = \underbrace{nm \cdot f(\alpha)}_{(V_3,U_2)} + \underbrace{\sum_{i=1}^{m} \beta e^{\alpha} n \cdot f\left(\frac{i}{m} \cdot \beta\right)}_{(V_1 \cup V_2, U_1)} = nm \cdot \left(f(\alpha) + e^{\alpha} \int_0^{\beta} f(x)\mathrm{d}x + o(1)\right), \quad (3.5)$$

where the second equality holds when $m$ goes to infinity.

Next, we analyze the the performance of Perturbed-Balance. We split the whole instance into three stages, corresponding to the arrivals of $V_1, V_2, V_3$ respectively.

**First stage ($V_1$).** Upon the arrival of each vertex in $V_1$, it matches uniformly to its neighbours in $U_1$. The behavior of Perturbed-Balance is the same for different groups. We analyze the matched portion of the last $\beta n$ vertices of each group after the first stage:

$$x_u = \frac{1}{\beta e^{\alpha} n} + \frac{1}{\beta e^{\alpha} n - 1} + \cdots + \frac{1}{\beta n} = \ln\left(\frac{\beta e^{\alpha} n}{\beta n}\right) + o(1) = \alpha + o(1),$$

where the equality holds when $n$ goes to infinity.

**Second Stage ($V_2$).** Upon the arrival of each vertex $v$ of $V_{2,i}$, it will be weighing the perturbed weights from $U_{1,i}$ and $U_2$:

$$f(x_{u_1}) \cdot w_{u_1} = f(\alpha + o(1)) \cdot f\left(\frac{i}{m}\right), \qquad \text{for } u_1 \in U_{1,i} \cap N(v),$$

$$\text{and} \quad f(x_{u_2}) \cdot w_{u_2} \geq f\left(\frac{i}{m}\right) \cdot f(\alpha), \qquad \text{for } u_2 \in U_2.$$

Notice that the perturbed weights from $U_2$ is always larger than the perturbed weights from $U_{1,i}$. We claim that in the second stage, all vertices of $V_2$ would be fully matched to $U_2$ by Perturbed-Balance. Thus, at the end of the second stage, all vertices in $U_2$ have matched portion $\beta$.

**Third stage ($V_3$).** The behavior of the last stage is similar to the behavior of the first stage, except that all vertices in $U_2$ start with a matched portion of $\beta$. After the arrival of the $k$-th vertex in $V_3$, the matched portion of its neighbor equals

$$\beta + \frac{1}{nm} + \frac{1}{nm-1} + \cdots + \frac{1}{nm-k+1} \geq \beta + \ln\left(\frac{nm}{nm-k+1}\right).$$

Consequently, only the first $(1 - e^{\beta-1})nm + 1$ vertices from $V_3$ can be matched. For the rest of the online vertices, all their neighbors would be fully matched before their arrivals.

To sum up, we calculate the performance of Perturbed-Balance.

$$\begin{aligned}\mathrm{ALG} &\leq \underbrace{\sum_{i=1}^{m} (\beta(e^{\alpha}-1)n + 1) \cdot f\left(\frac{i}{m} \cdot \beta\right)}_{(V_1,U_1)} + \underbrace{\beta nm \cdot f(\alpha)}_{(V_2,U_2)} + \underbrace{\left((1-e^{\beta-1})nm + 1\right) \cdot f(\alpha)}_{(V_3,U_2)} \\ &= nm \cdot \left((e^{\alpha}-1)\int_0^{\beta} f(x)\mathrm{d}x + (\beta + 1 - e^{\beta-1}) \cdot f(\alpha) + o(1)\right). \quad (3.6)\end{aligned}$$

Together with (3.5) and the assumption that Perturbed-Balance is $\Gamma$-competitive, we conclude the proof by letting $n, m \to \infty$:

$$(e^{\alpha}-1)\int_0^{\beta} f(x)\mathrm{d}x + (\beta+1-e^{\beta-1})\cdot f(\alpha) \geq \Gamma\cdot\left(f(\alpha)+e^{\alpha}\int_0^{\beta} f(x)\mathrm{d}x\right)$$

$$\iff \left(\beta+1-e^{\beta-1}-\Gamma\right)\cdot f(\alpha) \geq (1-(1-\Gamma)\cdot e^{\alpha})\cdot\int_0^{\beta} f(x)\mathrm{d}x, \quad \forall \alpha,\beta\in[0,1].$$

# 4 AdWords with Unknown Budget

In this section, we prove Theorem 2.

▶ **Theorem 2.** *The competitive ratio of* Perturbed-Ranking *algorithm with any perturbation function $f(x)$ on fractional AdWords is at most $1-1/e-0.0003$. In particular, using the canonical function $f(x)=1-e^{x-1}$, the competitive ratio is at most* 0.624.

We first construct a hard instance for which Perturbed-Ranking with $f(x)=1-e^{x-1}$ only achieves a competitive ratio of 0.624. Recall that the vertex-weighted online bipartite matching problem is a special case of the AdWords problem. Together with Theorem 1, it should be convincing that Perturbed-Ranking (with any perturbation function) cannot achieve the prominent competitive ratio of $1-1/e$.

Our construction for general perturbation functions has a similar structure as the construction for the canonical perturbation function. On the other hand, general perturbation functions introduce extra technical difficulties to our argument that we shall discuss soon.

## 4.1 Canonical Perturbation Function $f(x)=1-e^{x-1}$

We prove the result by the following lemma.

▶ **Lemma 12.** *If* Perturbed-Ranking *with perturbation function $f(x)=1-e^{x-1}$ achieves a competitive ratio of $\Gamma$ on AdWords, then*

$$(1-\Gamma)\cdot f(\alpha) \geq (\Gamma-\alpha)\cdot\int_0^{\alpha} f(x)\mathrm{d}x + \Gamma\cdot\int_{\alpha}^{1} f(x)\mathrm{d}x, \quad \forall\alpha\in[0,1]. \tag{4.1}$$

**Proof.** Fix $\alpha\in[0,1]$. Let $n$ be a sufficiently large number. Refer to Figure 4.1 for our instance.

Our construction consists of $n+1$ offline vertices $u_0,u_1,\ldots,u_n$ and $2n$ online vertices $v_1,v_2,\ldots,v_{2n}$. The budgets of $u_1,u_2,\ldots,u_n$ are all 1 and the budget of $u_0$ is unlimited. The online vertices arrive in ascending order of their indices, i.e. $v_i$ is the $i$-th arriving online vertex. Next, we define the edges of the graph:

- $v_1,v_2,\ldots,v_n$ are connected to $u_0$, with edge weights $b_1,b_2,\ldots,b_n$ respectively.
- $v_1,v_2,\ldots,v_{2n}$ are fully connected to $u_1,u_2,\ldots,u_n$ with edge weights 1.

Before we define the weights, we make an extra assumption to simplify our analysis:

$$\forall 1\leq i\leq n, \text{the rank of } u_i \text{ is } y_i = i/n.$$

Indeed, by Lemma 10, we would have that the set of ranks $\{y_1,\ldots,y_n\}$ are "close" to $\{\frac{1}{n},\frac{2}{n},\cdots,1\}$. Moreover, all vertices $u_1,\ldots,u_n$ are symmetric in our graph. This assumption would significantly simplify our analysis and can be removed by a more conservative choice of the edge weights. We omit the tedious details for simplicity.

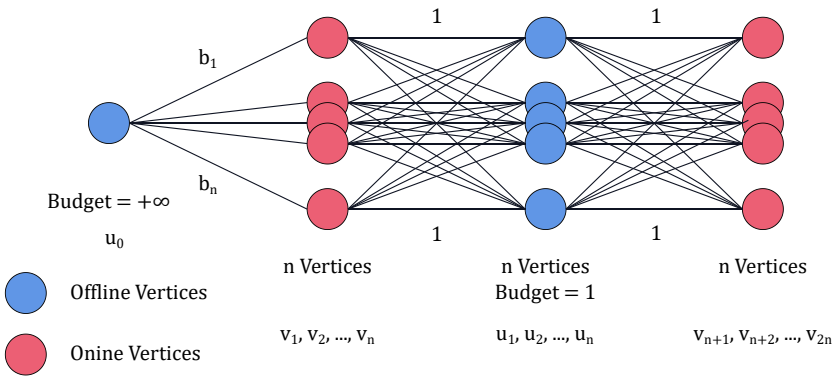



**Figure 4.1** Instance 2.

Let $b_i = \frac{f(i/n)}{f(\alpha)}$. The offline optimum is to match $v_1, v_2, \ldots, v_n$ to $u_0$ and to match $v_{n+1}, v_{n+2}, \ldots, v_{2n}$ to $u_1, u_2, \ldots, u_n$, respectively. Consequently,

$$\mathrm{OPT} = n + \sum_{i=1}^{n} b_i = n \cdot \left(1 + \frac{1}{f(\alpha)} \int_0^1 f(x)\mathrm{d}x + o(1)\right).$$

With the assumption, the only randomness of Perturbed-Ranking is the rank $y_0$ of $u_0$.

**Case 1.** $(y_0 \geq \alpha)$ For each online vertex $v_i$, the perturbed weight of $(u_0, v_i)$ is

$$b_i \cdot f(y_0) = \frac{f(y_0)}{f(\alpha)} \cdot f\left(\frac{i}{n}\right) \leq f\left(\frac{i}{n}\right),$$

while the perturbed weight of $(u_i, v_i)$ is $f(y_i) = f\left(\frac{i}{n}\right)$. Therefore, Perturbed-Ranking matches $(u_i, v_i)$ for all $1 \leq i \leq n$ and we have $\mathrm{ALG}(y_0) = n$.

**Case 2.** $(y_0 < \alpha)$ In this case, some of the $v_1, v_2, \ldots, v_n$ would be matched to $u_0$. However, the number of vertices matched to $u_0$ should be no more than $\alpha n$. The reason is as follows. For each online vertex $v_i$, suppose the number of previous vertices matched to $u_0$ is larger than $\alpha n$, then the perturbed weight of $(u_0, v_i)$ is $\frac{f(y_0)}{f(\alpha)} \cdot f\left(\frac{i}{n}\right)$, while the perturbed weight of $(u_{i-\alpha n}, v_i)$ is $f\left(\frac{i}{n} - \alpha\right)$. Notice that $f(x) = 1 - e^{x-1}$ is a log-concave function. Hence,

$$\frac{f(y_0)}{f(\alpha)} \cdot f\left(\frac{i}{n}\right) \leq \frac{f(0)}{f(\alpha)} \cdot f\left(\frac{i}{n}\right) \leq f\left(\frac{i}{n} - \alpha\right).$$

In other words, $v_i$ will not match $u_0$. This concludes the proof that the number of vertices matched to $u_0$ is no more than $\alpha n$. Notice that $b_i$'s are non-increasing, we have

$$\mathrm{ALG}(y_0) \leq \sum_{i=1}^{\alpha n} b_i + n = \sum_{i=1}^{\alpha n} \frac{f(i/n)}{f(\alpha)} + n = n \cdot \left(\frac{1}{f(\alpha)} \int_0^{\alpha} f(x)\mathrm{d}x + 1 + o(1)\right).$$

Putting the two cases together and assuming that Perturbed-Ranking algorithm is $\Gamma$-competitive, we conclude the proof of the lemma.

$$\Gamma \leq \frac{\mathbb{E}[\mathrm{ALG}]}{\mathrm{OPT}} = \frac{\alpha \cdot n \cdot \left(\frac{1}{f(\alpha)} \int_0^{\alpha} f(x)\mathrm{d}x + 1 + o(1)\right) + (1-\alpha) \cdot n}{n \cdot \left(1 + \frac{1}{f(\alpha)} \int_0^1 f(x)\mathrm{d}x + o(1)\right)} = \frac{\alpha \cdot \int_0^{\alpha} f(x)\mathrm{d}x + f(\alpha)}{\int_0^1 f(x)\mathrm{d}x + f(\alpha)}$$

$$\iff (1-\Gamma) \cdot f(\alpha) \geq (\Gamma - \alpha) \cdot \int_0^{\alpha} f(x)\mathrm{d}x + \Gamma \cdot \int_{\alpha}^1 f(x)\mathrm{d}x. \qquad \blacktriangleleft$$

▶ **Corollary 13.** Perturbed-Ranking *with* $f(x) = 1 - e^{x-1}$ *is at most* 0.624*-competitive for AdWords.*

**Proof.** Plugging in $\alpha = 0.1$ and $f(x) = 1 - e^{x-1}$ in equation (4.1), we have

$$\Gamma \le \frac{\alpha \cdot \int_0^\alpha f(x)\mathrm{d}x + f(\alpha)}{\int_0^1 f(x)\mathrm{d}x + f(\alpha)} = \frac{0.1 \cdot \left(0.1 - e^{-0.9} + e^{-1}\right) + 1 - e^{-0.9}}{e^{-1} + 1 - e^{-0.9}} < 0.624\ . \qquad \blacktriangleleft$$

## 4.2 General Perturbation Functions

Before we delve into the detailed proof, we explain the technical difficulty introduced by general perturbation functions. Our plan is to generalize Lemma 12: if we are able to prove equation (4.1) for an arbitrary function $f$, we would then conclude our theorem by combining it with equation (3.2).

However, our argument of the second case ($y_0 < \alpha$) of Lemma 12 crucially relies on the specific formula of $f(x)$. I.e., to upper bound the performance of Perturbed-Ranking, we use the fact that $f(0) \cdot f(x) \le f(\alpha) \cdot f(x - \alpha)$.

For a general perturbation function $f(x)$, if we stick to the same property that no more than $\alpha n$ vertices can be matched to $u_0$ when $y_0 < \alpha$, we could achieve it by setting the weights $b_i$ to be smaller. Indeed, if $f\left(\frac{i}{n} - \alpha\right) \ge f(0) \cdot b_i$ holds, the previous analysis can be easily generalized. Hence, a natural attempt is to modify the instance as the following.

$$b_i = \begin{cases} \frac{1}{f(\alpha)} \cdot f\left(\frac{i}{n}\right) & i < \alpha n, \\ \min\left\{\frac{1}{f(\alpha)} \cdot f\left(\frac{i}{n}\right), \frac{1}{f(0)} \cdot f\left(\frac{i}{n} - \alpha\right)\right\} & i \ge \alpha n. \end{cases}$$

Unfortunately, this modification is not strong enough to give a constant strictly smaller than $1 - 1/e$. The reason is that the function $f$ may have a steep drop in the neighborhood of 0, which leads to negligible $b_i$'s in the above construction.

On the other hand, the failure of the analysis comes from our coarse and brutal relaxation by establishing a single upper bound for all $y_0 < \alpha$. For instance, if the function steeply drops at some $\beta \in [0, \alpha]$, then we could resolve the issue by considering two cases of $y_0 < \beta$ or $y_0 \in [\beta, \alpha)$. Formally, we prove the following lemma that is slightly weaker than (4.1).

▶ **Lemma 14.** *If* Perturbed-Ranking *with perturbation function* $f(x)$ *achieves a competitive ratio of* $\Gamma$ *on AdWords, then*

$$(1 - \Gamma) \cdot f(\alpha) \ge (\Gamma - \alpha) \cdot \int_0^\alpha f(x)\mathrm{d}x + (\Gamma - \beta) \cdot \int_\alpha^1 \min\left\{f(x), \frac{f(\alpha)}{f(\beta)} f(x - \alpha)\right\}\mathrm{d}x, \quad \forall \alpha, \beta \in [0, 1], \alpha \ge \beta. \quad (4.2)$$

**Proof.** We apply the same construction as in Lemma 12 and make the same assumption that $y_i = \frac{i}{n}$ for the simplicity of presentation. We modify the instance by setting the weights $b_i$ differently:

$$\forall 1 \le i \le n, \quad b_i = \begin{cases} \frac{1}{f(\alpha)} \cdot f\left(\frac{i}{n}\right) & i < \alpha n, \\ \min\left\{\frac{1}{f(\alpha)} \cdot f\left(\frac{i}{n}\right), \frac{1}{f(\beta)} \cdot f\left(\frac{i}{n} - \alpha\right)\right\} & i \ge \alpha n. \end{cases}$$

The optimal solution equals

$$\mathrm{OPT} = n + \sum_{i=1}^{n} b_i = n \cdot \left(1 + \frac{1}{f(\alpha)} \int_0^\alpha f(x)\mathrm{d}x + \int_\alpha^1 \min\left\{\frac{f(x)}{f(\alpha)}, \frac{f(x - \alpha)}{f(\beta)}\right\}\mathrm{d}x + o(1)\right). \quad (4.3)$$

Next, we consider the performance of Perturbed-Ranking depending on the value of $y_0$.

**Case 1.** $(y_0 < \beta)$ We use OPT as a trivial upper bound of ALG, i.e., $\mathrm{ALG}(y_0) \le \mathrm{OPT}$.

**Case 2.** $(y_0 \ge \alpha)$ For each online vertex $v_i$, the perturbed weight of $(u_0, v_i)$ is

$$b_i \cdot f(y_0) \le \frac{f(y_0)}{f(\alpha)} \cdot f\left(\frac{i}{n}\right) \le f\left(\frac{i}{n}\right),$$

while the perturbed weight of $(u_i, v_i)$ is $f(y_i) = f\left(\frac{i}{n}\right)$. Therefore, Perturbed-Ranking matches $(u_i, v_i)$ for all $1 \le i \le n$ and we have $\mathrm{ALG}(y_0) = n$.

**Case 3.** $(y_0 \in [\beta, \alpha))$ We prove that the number of vertices matched to $u_0$ is no more than $\alpha n$. This can be similarly argued as follows. For each online vertex $i$ $(i > \alpha n)$, suppose the number of vertices matched to $u_0$ is already $\alpha n$, the perturbed weight for $u_0$ is $f(y_0) \cdot b_i \le \frac{f(y_0)}{f(\beta)} \cdot f\left(\frac{i}{n} - \alpha\right)$, while the perturbed weight for $u_{i-\alpha n}$ is $f\left(\frac{i}{n} - \alpha\right)$. So that $f\left(\frac{i}{n} - \alpha\right) \ge f(y_0) \cdot b_i$ and thus $v_i$ will not choose $u_0$ again. Therefore, as the number of vertices matched to $u_0$ is no more than $\alpha n$, we have

$$\mathrm{ALG} \le n + \sum_{i=1}^{\alpha n} b_i = n \cdot \left(1 + \frac{1}{f(\alpha)} \int_0^{\alpha} f(x)\mathrm{d}x + o(1)\right).$$

Taking expectation over the randomness of $y_0$, we conclude that

$$\mathbb{E}[\mathrm{ALG}] \le n \cdot \left(\frac{\alpha - \beta}{f(\alpha)} \int_0^{\alpha} f(x)\mathrm{d}x + 1 - \beta + o(1)\right) + \beta \cdot \mathrm{OPT}. \tag{4.4}$$

Finally, we conclude the proof by plugging in (4.3) and (4.4) to $\mathbb{E}[\mathrm{ALG}] \ge \Gamma \cdot \mathrm{OPT}$ and letting $n$ goes to infinity. ◀

Recall that the vertex-weighted online bipartite matching problem is a special case of AdWords. We conclude the proof of Theorem 2 by Lemma 11, 14 and the following mathematical fact. We provide the proof of the following lemma in the full version.

▶ **Lemma 15.** *If a perturbation function $f$ and $\Gamma > 0$ satisfy the following conditions:*

$$\left(\alpha + 1 - e^{\alpha - 1} - \Gamma\right) \cdot f(\beta) \ge \left(1 - (1 - \Gamma) \cdot e^{\beta}\right) \cdot \int_0^{\alpha} f(x)\mathrm{d}x, \quad \forall \alpha, \beta \in [0, 1], \tag{4.5}$$

$$(1 - \Gamma) f(\alpha) \ge (\Gamma - \alpha) \int_0^{\alpha} f(x)\mathrm{d}x + (\Gamma - \beta) \int_{\alpha}^{1} \min\left\{f(x), \frac{f(\alpha)}{f(\beta)} f(x - \alpha)\right\}\mathrm{d}x, \quad \forall \alpha, \beta \in [0, 1], \alpha \ge \beta, \tag{4.6}$$

*then* $\Gamma < 1 - 1/e - 0.0003$.

## References


1 Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *SODA*, pages 1253–1264, 2011. URL: `http://www.siam.org/proceedings/soda/2011/SODA11_094_aggarwalg.pdf`.

2 Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *ACM SIGACT News*, 39(1):80–87, 2008.

3 Guy Blanc and Moses Charikar. Multiway online correlated selection. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1277–1284. IEEE, 2021. `doi:10.1109/FOCS52979.2021.00124`.

4 Niv Buchbinder, Moran Feldman, Yuval Filmus, and Mohit Garg. Online submodular maximization: beating 1/2 made simple. *Math. Program.*, 183(1):149–169, 2020. doi:10.1007/s10107-019-01459-z.

5 T.-H. Hubert Chan, Fei Chen, Xiaowei Wu, and Zhichao Zhao. Ranking on arbitrary graphs: Rematch via continuous linear programming. *SIAM J. Comput.*, 47(4):1529–1546, 2018.

6 Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In John Chuang, Lance Fortnow, and Pearl Pu, editors, *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 71–78. ACM, 2009. doi:10.1145/1566374.1566384.

7 Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *SODA*, pages 101–107. SIAM, 2013.

8 Alon Eden, Michal Feldman, Amos Fiat, and Kineret Segal. An economics-based analysis of RANKING for online bipartite matching. In *SOSA*, pages 107–110. SIAM, 2021.

9 Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. In *FOCS*, pages 412–423. IEEE, 2020.

10 Ruiquan Gao, Zhongtian He, Zhiyi Huang, Zipei Nie, Bijun Yuan, and Yan Zhong. Improved online correlated selection. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1265–1276. IEEE, 2021. doi:10.1109/FOCS52979.2021.00123.

11 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, pages 982–991, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347189.

12 Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. Fully online matching. *J. ACM*, 67(3):17:1–17:25, 2020.

13 Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *SODA*, pages 2875–2886. SIAM, 2019.

14 Zhiyi Huang and Xinkai Shu. Online stochastic matching, poisson arrivals, and the natural linear program. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 682–693. ACM, 2021. doi:10.1145/3406325.3451079.

15 Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals. *ACM Transactions on Algorithms*, 15(3):1–15, 2019.

16 Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Fully online matching II: beating ranking and water-filling. In *FOCS*, pages 1380–1391. IEEE, 2020.

17 Zhiyi Huang, Qiankun Zhang, and Yuhao Zhang. Adwords in a panorama. In *FOCS*, pages 1416–1426. IEEE, 2020.

18 Billy Jin and David P. Williamson. Improved analysis of RANKING for online vertex-weighted bipartite matching in the random order model. In Michal Feldman, Hu Fu, and Inbal Talgam-Cohen, editors, *Web and Internet Economics - 17th International Conference, WINE 2021, Potsdam, Germany, December 14-17, 2021, Proceedings*, volume 13112 of *Lecture Notes in Computer Science*, pages 207–225. Springer, 2021. doi:10.1007/978-3-030-94676-0_12.

19 Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233(1-2):319–325, 2000.

20 Michael Kapralov, Ian Post, and Jan Vondrák. Online submodular welfare maximization: Greedy is optimal. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1216–1225. SIAM, 2013. doi:10.1137/1.9781611973105.88.

21 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *STOC*, pages 587–596, 2011. doi:10.1145/1993636.1993715.

22 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358, 1990. doi:10.1145/100216.100262.

23 Nitish Korula, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM J. Comput.*, 47(3):1056–1086, 2018. doi:10.1137/15M1051142.

24 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *STOC*, pages 597–606, 2011. doi:10.1145/1993636.1993716.

25 Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

26 Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Towards a better understanding of randomized greedy matching. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1097–1110. ACM, 2020. doi:10.1145/3357713.3384265.

27 Rajan Udwani. Adwords with unknown budgets. *CoRR*, abs/2110.00504, 2021. arXiv:2110.00504.

28 Vijay V. Vazirani. Online bipartite matching and adwords. *CoRR*, abs/2107.10777, 2021. arXiv:2107.10777.

# Tight Bounds for Quantum Phase Estimation and Related Problems

**Nikhil S. Mande**
University of Liverpool, UK

**Ronald de Wolf**
QuSoft, CWI and University of Amsterdam, The Netherlands

## Abstract

Phase estimation, due to Kitaev [arXiv'95], is one of the most fundamental subroutines in quantum computing, used in Shor's factoring algorithm, optimization algorithms, quantum chemistry algorithms, and many others. In the basic scenario, one is given black-box access to a unitary $U$, and an eigenstate $|\psi\rangle$ of $U$ with unknown eigenvalue $e^{i\theta}$, and the task is to estimate the eigenphase $\theta$ within $\pm\delta$, with high probability. The repeated application of $U$ and $U^{-1}$ is typically the most expensive part of phase estimation, so for us the *cost* of an algorithm will be that number of applications.

Motivated by the "guided Hamiltonian problem" in quantum chemistry, we tightly characterize the cost of several variants of phase estimation where we are no longer given an arbitrary eigenstate, but are required to estimate the *maximum* eigenphase of $U$, aided by *advice* in the form of states (or a unitary preparing those states) which are promised to have at least a certain overlap $\gamma$ with the top eigenspace. We give algorithms and matching lower bounds (up to logarithmic factors) for all ranges of parameters. We show a crossover point below which advice is not helpful: $o(1/\gamma^2)$ copies of the advice state (or $o(1/\gamma)$ applications of an advice-preparing unitary) are not significantly better than having no advice at all. We also show that having knowledge of the eigenbasis of $U$ does not significantly reduce cost. Our upper bounds use the subroutine of *generalized maximum-finding* of van Apeldoorn, Gilyén, Gribling, and de Wolf [Quantum'20], the state-based Hamiltonian simulation of Lloyd, Mohseni, and Rebentrost [Nature Physics'13], and several other techniques. Our lower bounds follow by reductions from a *fractional* version of the Boolean OR function *with advice*, which we lower bound by a simple modification of the adversary method of Ambainis [JCSS'02]. As an immediate consequence we also obtain a lower bound on the complexity of the Unitary recurrence time problem, matching an upper bound of She and Yuen [ITCS'23] and resolving an open question posed by them.

Lastly, we study how efficiently one can reduce the error probability in the basic phase-estimation scenario. We show that an algorithm solving phase estimation to precision $\delta$ with error probability at most $\varepsilon$ must have cost $\Omega\left(\frac{1}{\delta}\log\frac{1}{\epsilon}\right)$, matching the obvious way to error-reduce the basic constant-error-probability phase estimation algorithm. This contrasts with some other scenarios in quantum computing (e.g. search) where error-reduction costs only a factor $O(\sqrt{\log(1/\epsilon)})$. Our lower bound technique uses a variant of the polynomial method with trigonometric polynomials.

**2012 ACM Subject Classification** Theory of computation → Oracles and decision trees; Theory of computation → Quantum complexity theory

**Keywords and phrases** Phase estimation, quantum computing

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.81

**Related Version** *Full Version*: https://arxiv.org/abs/2305.04908 [25]

**Funding** *Nikhil S. Mande*: Part of this work was done while the author was a postdoc at QuSoft and CWI, Amsterdam.
*Ronald de Wolf*: Partially supported by the Dutch Research Council (NWO/OCW), as part of the Quantum Software Consortium programme (project number 024.003.037).

**Acknowledgements** We thank Jordi Weggemans for useful comments and for a pointer to [21].





# 1 Introduction

## 1.1 Phase estimation

Kitaev [19] gave an elegant and efficient quantum algorithm for the task of *phase estimation* nearly 30 years ago. The task is easy to state: given black-box access to a unitary and an eigenstate, estimate the phase of the associated eigenvalue. Roughly speaking, the standard algorithm for this task sets up a superposition involving many different powers of the unitary to extract many different powers of the eigenvalue, and then uses a quantum Fourier transform to turn that into an estimate of the eigenphase.[1] Many of the most prominent quantum algorithms can either be phrased as phase estimation, or use phase estimation as a crucial subroutine. Some examples are Shor's period-finding algorithm [30] as presented in [10]; approximate counting [6] can be done using phase estimation on the unitary of one iteration of Grover's search algorithm [16], which also recovers the $O(\sqrt{N})$ complexity for searching an $N$-element unordered search space; the HHL algorithm for solving linear systems of equations estimates eigenvalues in order to invert them [17]. Applications of phase estimation in quantum chemistry are also very prominent, as discussed below.

More precisely, we are given black-box access to an $N$-dimensional unitary $U$ (and a controlled version thereof) and a state $|\psi\rangle$ that satisfies $U|\psi\rangle = e^{i\theta}|\psi\rangle$. Our goal is to output (with probability at least $2/3$) a $\tilde{\theta} \in [0, 2\pi)$ such that $|\tilde{\theta} - \theta|$ is at most $\delta$ in $\mathbb{R}$ mod $2\pi$. In the basic scenario we are given access to one copy of $|\psi\rangle$, and are allowed to apply $U$ and its inverse. Since the repeated applications of $U$ and $U^{-1}$ are typically the most expensive parts of algorithms for phase estimation, the *cost* we wish to minimize is the number of applications of $U$ and $U^{-1}$. We are additionally allowed arbitrary unitaries that do not depend on $U$, at no cost. Kitaev's algorithm has cost $O(1/\delta)$.

## 1.2 Phase estimation with advice

One of the core problems in quantum chemistry is the following: given a classical description of some Hamiltonian $H$ (for instance an "electronic structure" Hamiltonian in the form of a small number of local terms), estimate its *ground state energy*, which is its smallest eigenvalue. If $H$ is normalized such that its eigenvalues are all in $[0, 2\pi)$ and we define the unitary $U = e^{iH}$ (which has the same eigenvectors as $H$, with an eigenvalue $\lambda$ of $H$ becoming the eigenvalue $e^{i\lambda}$ for $U$), then finding the ground state energy of $H$ is equivalent to finding the smallest eigenphase of $U$. If we are additionally given a *ground state* $|\psi\rangle$ (i.e., an eigenstate corresponding to the smallest eigenphase), then phase estimation is tailor-made to estimate the ground state energy. However, in quantum chemistry it is typically hard to prepare the ground state of $H$, or even something close to it. What can sometimes be done is the preparation of some quantum state that has some non-negligible "overlap" $\gamma$ with the ground space, for instance the "Hartree-Fock state". We will call such a state an *advice state*. In the complexity-theoretic context, this problem of ground state estimation for a local Hamiltonian given an advice state, is known as the "guided local Hamiltonian problem", and has received quite some attention recently [13, 8, 12, 32] because of its connections with quantum chemistry as well as deep complexity questions such as the PCP conjecture. These complexity-theoretic results typically focus on the BQP-completeness of certain special cases

[1] An added advantage of the standard algorithm for phase estimation is that it can also work with a quantum Fourier transform that is correct on average rather than in the worst case [23]. However, there are also approaches to phase estimation that avoid the QFT altogether, see e.g. [28].

of the guided local Hamiltonian problem, and don't care about polynomial overheads of the cost in the number of qubits $\log N$ and in the parameters $\delta$ and $\gamma$. In contrast, we care here about getting essentially optimal bounds on the cost of phase estimation in various scenarios.

To be more precise, suppose our input unitary is $U = \sum_{j=0}^{N-1} e^{i\theta_j}|u_j\rangle\langle u_j|$ with each $\theta_j \in [0, 2\pi)$. Let $\theta_{\max} = \max_{j\in\{0,1,\ldots,N-1\}} \theta_j$ denote the maximum eigenphase, and let $S$ denote the space spanned by all eigenstates with eigenphase $\theta_{\max}$, i.e., the "top eigenspace". Advice is given in the form of a state $|\alpha\rangle$ whose projection on $S$ has squared norm at least $\gamma^2$: $\|P_S|\alpha\rangle\|^2 \geq \gamma^2$. Note that if $S$ is spanned by a single eigenstate $|u_{\max}\rangle$, then this condition is the same as $|\langle\alpha|u_{\max}\rangle| \geq \gamma$, which is why we call $\gamma$ the *overlap* of the advice state with the target eigenspace. The task $\mathsf{maxQPE}_{N,\delta}$ is to output, with probability at least 2/3, a $\delta$-precise (in $\mathbb{R}$ mod $2\pi$) estimate of $\theta_{\max}$.[2]

We will distinguish between the setting where the advice is given in the form of a number of copies of the advice state $|\alpha\rangle$, or the potentially more powerful setting where we can apply (multiple times) a *unitary* $A$ that prepares $|\alpha\rangle$ from some easy-to-prepare initial state, say $|0\rangle$. We would have such a unitary $A$ for instance if we have a procedure to prepare $|\alpha\rangle$ ourselves in the lab. We can also distinguish between the situation where the eigenbasis $|u_0\rangle, \ldots, |u_N\rangle$ of $U$ is known (say, the computational basis where $|u_j\rangle = |j\rangle$) and the potentially harder situation where the eigenbasis is unknown. These two binary distinctions give us four different settings. For each of these settings we determine essentially optimal bounds on the cost of phase estimation, summarized in Table 1.

**Table 1** Our results for the cost of $\mathsf{maxQPE}_{N,\delta}$. We assume $\gamma > 1/\sqrt{N}$ since a random state has overlap $1/\sqrt{N}$ with the target eigenspace with high probability, and such a state can be prepared at no cost. The "Basis" column indicates whether the eigenbasis of $U$ is known; "Access to advice" indicates whether we get copies of the advice state or a unitary to prepare it; "Number of accesses" refers to the number of accesses to advice that we have. The last two columns show our bounds with references to the lemmas where they are stated and proved. The $\widetilde{O}(\cdot)$ in the upper-bound column hides a factor $\log N$ for the odd-numbered rows, and $\log(1/\gamma)$ for the even-numbered rows.

| Row | Basis | Access to advice | Number of accesses | Upper bound | Lower bound |
|---|---|---|---|---|---|
| 1 | known | state | $o\left(\frac{1}{\gamma^2}\right)$ | $\widetilde{O}\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 20 | $\Omega\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 13 |
| 2 | known | state | $\Omega\left(\frac{1}{\gamma^2}\right)$ | $\widetilde{O}\left(\frac{1}{\gamma\delta}\right)$, Lemma 22 | $\Omega\left(\frac{1}{\gamma\delta}\right)$, Lemma 14 |
| 3 | unknown | state | $o\left(\frac{1}{\gamma^2}\right)$ | $\widetilde{O}\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 20 | $\Omega\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 13 |
| 4 | unknown | state | $\Omega\left(\frac{1}{\gamma^2}\right)$ | $\widetilde{O}\left(\frac{1}{\gamma\delta}\right)$, Lemma 22 | $\Omega\left(\frac{1}{\gamma\delta}\right)$, Lemma 14 |
| 5 | known | unitary | $o\left(\frac{1}{\gamma}\right)$ | $\widetilde{O}\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 20 | $\Omega\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 15 |
| 6 | known | unitary | $\Omega\left(\frac{1}{\gamma}\right)$ | $\widetilde{O}\left(\frac{1}{\gamma\delta}\right)$, Lemma 21 | $\Omega\left(\frac{1}{\gamma\delta}\right)$, Lemma 16 |
| 7 | unknown | unitary | $o\left(\frac{1}{\gamma}\right)$ | $\widetilde{O}\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 20 | $\Omega\left(\frac{\sqrt{N}}{\delta}\right)$, Lemma 15 |
| 8 | unknown | unitary | $\Omega\left(\frac{1}{\gamma}\right)$ | $\widetilde{O}\left(\frac{1}{\gamma\delta}\right)$, Lemma 21 | $\Omega\left(\frac{1}{\gamma\delta}\right)$, Lemma 16 |

Let us highlight some interesting consequences of our results. First, a little bit of advice is no better than no advice: the upper bounds in the odd-numbered rows of Table 1 are actually obtained by algorithms that don't use the given advice ($o(1/\gamma^2)$ copies of $|\alpha\rangle$ or $o(1/\gamma)$ applications of $A$ and $A^{-1}$) at all, yet their costs essentially match the lower bounds for algorithms that use advice.

[2] It doesn't really matter, but we focus on the *maximum* rather than minimum eigenphase of $U$ because eigenphase 0 (i.e., eigenvalue 1) is a natural baseline, and we are looking for the eigenphase furthest away from this baseline.

We remark here that the same proofs yield the same asymptotic lower bounds for algorithms with access to at most $c/\gamma^2$ advice states for Theorem 12, Rows 1 and 3 of Table 1, and for algorithms with access to at most $c/\gamma$ advice unitaries for Rows 5 and 7 of Table 1, where $c$ is a suitably small constant. We chose to use $o(\cdot)$ to avoid clutter.

A second interesting consequence is that too much advice is no better than a moderate amount of advice: the upper bounds in Rows 2 and 4 use $O(1/\gamma^2)$ advice states, and the upper bounds in Rows 6 and 8 use $O(1/\gamma)$ advice unitaries, and using more advice does not reduce the cost further. Thirdly, it turns out that knowledge of the eigenbasis of $U$ doesn't really help in reducing the cost: the costs in row 1 and row 3 are the same, and similarly for rows 2 vs. 4, 5 vs. 7 and 6 vs. 8.

Our upper bounds use the subroutine of *generalized maximum-finding* of van Apeldoorn, Gilyén, Gribling, and de Wolf [2] which allows us to find maximum values in the second register of a two-register superposition even when the first of these two registers has an unknown basis. We derive the upper bound of row 4 from the upper bound of row 8 by using roughly $1/\gamma$ copies of $|\alpha\rangle$ to simulate one reflection around the state $|\alpha\rangle = A|0\rangle$, using the techniques of Lloyd, Mohseni, and Rebentrost [24].[3] Our lower bounds follow from reductions from a fractional version of the Boolean OR function with advice. We show a lower bound for this by a simple modification of the adversary method [1] taking into account the input-dependent advice in the initial state.

### Comparison with related work

Some of the results in our table were already (partially) known. A cost-$\widetilde{O}(\sqrt{N}/\delta)$ algorithm for the adviceless setting with unknown eigenbasis (implying the upper bounds of rows 1, 3, 5, 7) was originally due to Poulin and Wocjan [27], and subsequently improved in the log-factors by van Apeldoorn et al. [2]; the latter algorithm is basically our proof of Lemma 20. Lin and Tong [21] (improving upon [11]) studied the situation with an advice-preparing unitary. Their setting is slightly different from ours, they focus on preparing the ground state[4] of a given Hamiltonian without a known bound on its spectrum, but [21, Theorem 8] implies a cost-$O(\log(1/\gamma)\log(1/\delta)\log\log(1/\delta)/\gamma\delta)$ algorithm for our row 8. Their follow-up paper [22] further reduces the number of auxiliary qubits with a view to near-term implementation, but does not reduce the cost further. Our cost-$O(\log(1/\gamma)/\gamma\delta)$ algorithm is slightly better in the log-factors than theirs, and uses quite different techniques ([21] uses quantum singular value transformation [15]).

On the lower-bound side, $\Omega(1/\delta)$ for the cost of phase estimation has long been known to hold when the success probability is required to be a constant, this follows for instance from the approximate counting lower bound of Nayak and Wu [26] (see also [4]). Lin and Tong [21, Theorem 10] proved lower bounds of $\Omega(1/\gamma)$ and $\Omega(1/\delta)$ on the cost for the setting with known eigenbasis and advice unitary (our row 6, and hence also row 8). This is subsumed by our stronger (and essentially optimal) $\Omega(1/\gamma\delta)$ lower bound in row 6. As far as we are aware, ours is the first paper to systematically tie together these different results and to complete the table with tight upper and lower bounds for the cost in all 8 cases.

---

[3] We only stated the cost (number of applications of $U$ and $U^{-1}$) of our algorithms here in the upper-bound column of Table 1. However, one can verify that the gate-complexities of our algorithms are only worse by log-factors: they use three main subroutines, all of which have only small overheads in gate-complexity. These subroutines are basic quantum phase estimation [19], generalized maximum-finding [2], and the simulation of a unitary reflecting about the state $|\alpha\rangle$ given a small number of copies of $|\alpha\rangle$.

[4] Because generalized maximum-finding (Lemma 17) actually outputs a state in addition to an estimate, our algorithms can be modified to also output a state that is close to the top eigenspace of $U$.

Let us also mention some recent work that is not directly covered by our results. First, lower bounds for the slightly unusual small-success-probability regime were recently studied by Lin [20]. Second, there has been work to make phase estimation more efficient in the important special case where the unitary $U = e^{iH}$ is induced by a Hamiltonian $H$ given classically as the sum of relatively simple terms, when the cost of phase estimation interacts with the cost of Hamiltonian simulation. See for instance the recent paper by Wan, Berta, and Campbell [31] and references therein.

#### Application

She and Yuen [29, Theorems 1.6 and 1.7] studied the $(t, \delta)$-*Unitary recurrence time problem*, which is to distinguish whether an input unitary $U$ satisfies $U^t = I$ or $\|U^t - I\| \geq \delta$, promised that one of these is the case (see Definition 7). They proved non-matching upper and lower bounds for the cost of quantum algorithms for this problem (see Theorem 8 in this paper). As an immediate application of our lower bound for fractional OR with advice, we also obtain improved lower bounds for the unitary recurrence time problem that match the upper bound of She and Yuen and answer one of their open problems [29, Section 2].

▶ **Theorem 1** (Lower bound for Unitary recurrence time). *Any quantum algorithm solving the $(t, \delta)$-recurrence time problem for $N$-dimensional unitaries has cost $\Omega(t\sqrt{N}/\delta)$.*

Interestingly, our lower bound uses the adversary method as opposed to their usage of the polynomial method.

### 1.3 Phase estimation with small error probability

For our results in this subsection we revert to the original scenario of phase estimation, where an algorithm is given the actual eigenstate $|\psi\rangle$ as input and the goal is to estimate its eigenphase $\theta$. However, we now consider the regime where we want small error probability $\varepsilon$ rather than constant error probability $1/3$. Let $\mathsf{QPE}_{N,\delta,\varepsilon}$ denote the task of computing, with error probability $\leq \varepsilon$, a $\delta$-approximation of $\theta$. By repeating Kitaev's $O(1/\delta)$-cost phase estimation algorithm $O(\log(1/\varepsilon))$ times and taking the median of the answers, we have the following $\varepsilon$-dependent upper bound.

▶ **Theorem 2** (Kitaev + standard error-reduction). *For all integers $N \geq 2$ and all $\varepsilon \in (0, 1/2), \delta \in [0, 2\pi)$, there exists an algorithm that solves $\mathsf{QPE}_{N,\delta,\varepsilon}$ with cost $O\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)$.*

Grover's algorithm [16] can compute the $\mathsf{OR}_N$ function with error probability $\leq 1/3$ using $O(\sqrt{N})$ queries to its $N$ input bits. Interestingly, there exists an $\varepsilon$-error quantum algorithm for $\mathsf{OR}_N$ with only $O(\sqrt{N\log(1/\varepsilon)})$ queries, which is asymptotically optimal [7], and similarly one can reduce error from $1/3$ to $\varepsilon$ for all symmetric Boolean functions at the expense of only a factor $\sqrt{\log(1/\varepsilon)}$ in the query complexity [33]. This is a speed-up over the naive $O(\log(1/\varepsilon))$ multiplicative overhead. Since optimal quantum algorithms with error probability $1/3$ for $\mathsf{OR}_N$ and for all symmetric functions can be derived from phase estimation, one may ask if one can achieve such an efficient error-reduction for quantum phase estimation as well: is there an algorithm for $\mathsf{QPE}_{N,\delta,\varepsilon}$ of cost $O\left(\frac{1}{\delta}\sqrt{\log(1/\varepsilon)}\right)$? We answer this in the negative, showing Theorem 2 is tight.

▶ **Theorem 3.** *For integers $N \geq 2$ and $\varepsilon, \delta \in (0, 1/2)$,*[5] *every algorithm that solves* $\mathsf{QPE}_{N,\delta,\varepsilon}$ *has cost* $\Omega\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)$.

In particular, this means that the optimal complexity of $\mathsf{OR}_N$ with small error probability $\varepsilon$ of [7] can*not* be derived from a phase estimation routine, in contrast to the case of $\mathsf{OR}_N$ (and search) with constant error probability. To show Theorem 3 we first argue that a cost-$C$ algorithm for $\mathsf{QPE}_{N,\delta,\varepsilon}$ gives us a cost-$C$ algorithm that distinguishes $U = I$ versus $U = I - (1 - e^{i\theta})|0\rangle\langle 0|$ where $\theta \notin [-3\delta, 3\delta] \bmod 2\pi$. We then note that the acceptance probability of such an algorithm can be written as a degree-$2C$ *trigonometric* polynomial in $\theta$, and invoke a known upper bound on the growth of such trigonometric polynomials in order to lower bound their degree.

## 2 Preliminaries

We state the required preliminaries in this section. All logarithms are taken base 2. For a positive integer $N$, $U(N)$ denotes the space of $N$-dimensional unitaries, and $I$ denote the $N$-dimensional Identity matrix (we drop the subscript if the dimension is clear from context).

For a positive integer $N \geq 2$ and a value $\theta \in [0, 2\pi)$, define the $N$-dimensional unitary $U_\theta$ as $U_\theta = I - (1 - e^{i\theta})|0\rangle\langle 0|$. In other words, $U_\theta$ is the diagonal matrix with all 1's except the first entry, which is $e^{i\theta}$. For an integer $j \in \{0, 1, \ldots, N-1\}$ and $\delta \in [0, 2\pi)$, define $M_{j,\delta} = I - (1 - e^{i\delta})|j\rangle\langle j|$.

### 2.1 Model of computation

Here we give a description of our model of computation for all tasks considered in this paper. All problems considered in this paper have the following properties:

- **Input:** An $N$-dimensional unitary $U$. We have access to the input as described below.
- **State space:** The state space of an algorithm comprises two registers: the first register is $N$-dimensional, and the second register is an arbitrarily large workspace.
- **Access to input and allowed operations:** An algorithm $\mathcal{A}$ may apply $U$ and $U^{-1}$ to the first register, and unitaries independent of $U$ to the whole space. It performs a POVM at the end to determine the classical output.
- **Cost of an algorithm:** Total number of applications of $U$ and $U^{-1}$.

Depending on the specific problem under consideration, the following properties are variable.

- **Initial state:** The initial state is assumed to be $|0\rangle|0\rangle$ unless mentioned otherwise.
- **Input promise:** The subset of $U(N)$ (possibly the full set) from which the input is taken.
- **Output:** The output requirement.
- **Advice:** We may be given access to a specific number of "advice states" $|\alpha\rangle$, or access to a specific number of applications of a unitary $A$ that prepares an advice state (e.g., $A|0\rangle = |\alpha\rangle$).

### 2.2 Problems of interest

We list our problems of interest here. All problems fit in the framework of the previous subsection, so we skip descriptions of the input, access to the input and allowed operations, and the workspace.

[5] We require $\delta < 2\pi/5$ for our proof of Claim 23 to work. This requirement can be strengthened a little to $\delta < 2\pi/3$, but we state our theorem with $\delta < 1/2$ for ease of notation.

▶ **Definition 4** (Phase Estimation). *Let $N \geq 2$ be an integer and $\varepsilon, \delta > 0$. The task* $\mathsf{QPE}_{N,\delta,\varepsilon}$ *is:*

- ***Advice:*** *We are given a single state $|\psi\rangle$ (in other words, our starting state is $|\psi\rangle|0\rangle$) with the promise that $U|\psi\rangle = e^{i\theta}|\psi\rangle$.*
- ***Output:*** *With probability at least $1-\varepsilon$, output $\tilde{\theta} \in [0, 2\pi)$ such that $|\tilde{\theta} - \theta| \leq \delta \mod 2\pi$.*

▶ **Definition 5.** *Let $N \geq 2$ be an integer and $\varepsilon, \delta \in (0,1)$. The task* $\mathsf{dist}_{N,\delta,\varepsilon}$ *is:*

- ***Input promise:*** *$U \in \{I, \{U_\theta : \theta \notin [\delta, \delta] \mod 2\pi\}\}$.*
- ***Output:*** *With probability at least $1-\varepsilon$, output 1 if $U = I$, and output 0 otherwise.*

We next define the natural variant of phase estimation that we consider when an algorithm need not be given a state from the target eigenspace.

▶ **Definition 6** (Maximum phase estimation). *Let $N \geq 2$ be an integer and $\delta > 0$. The task* $\mathsf{maxQPE}_{N,\delta}$ *is:*

- ***Input promise:*** *We consider two cases: one where the eigenbasis of $U$ is known, and the other where it is unknown. In the former case, we may assume $U = \sum_{j=0}^{N-1} e^{i\theta_j}|j\rangle\langle j|$. Define $\theta_{\max} = \max_{j \in \{0,1,\ldots,N-1\}} \theta_j \in [0, 2\pi)$.*
- ***Advice:*** *We consider two cases:*
  - *In one case we are given access to advice in the form of a state $|\alpha\rangle$ such that $\|P_S|\alpha\rangle\|^2 \geq \gamma^2$, where $P_S$ denotes the projection on $S$, the space of all eigenstates with eigenphase $\theta_{\max}$. If $S$ is spanned by one $|u_{\max}\rangle$, this requirement is the same as $|\langle\alpha|u_{\max}\rangle| \geq \gamma$.*
  - *In the other case, we have black-box access to a unitary $A$ that prepares such a state $|\alpha\rangle$. We can apply $A$ and $A^{-1}$. As before, $\gamma$ is the* overlap *of $|\alpha\rangle$ with the target eigenspace.*
- ***Number of accesses to advice:*** *We either have 'few' accesses to advice as defined above ($o(1/\gamma^2)$ advice states or $o(1/\gamma)$ advice unitaries), or 'many' accesses to advice ($\Omega(1/\gamma^2)$ advice states or $\Omega(1/\gamma)$ advice unitaries).*
- ***Output:*** *With probability at least $2/3$, output a value in $[\theta_{\max} - \delta, \theta_{\max} + \delta] \mod 2\pi$.*

▶ **Definition 7** (Unitary recurrence time, [29, Definition 1.5]). *For integers $N \geq 2, t \geq 1$ and $\delta \in (0,1)$, the $(t,\delta)$-recurrence time problem is:*

- ***Input promise:*** *Either $U = I$, or $\|U^t - I\| \geq \delta$ in spectral norm.*
- ***Output:*** *With probability at least $2/3$: output 1 if $U = I$, and 0 otherwise.*

The following are the non-matching upper and lower bounds for this problem of She and Yuen [29].

▶ **Theorem 8** ([29, Theorems 1.6 and 1.7]). *Let $\delta \leq \frac{1}{2\pi}$. Every quantum algorithm solving the $(t,\delta)$-recurrence time problem for $d$-dimensional unitaries has cost $\Omega\left(\max\left(t/\delta, \sqrt{d}\right)\right)$. The $(t,\delta)$-recurrence time problem can be solved with cost $O(t\sqrt{d}/\delta)$.*

## 2.3 Trigonometric polynomials and their growth

▶ **Definition 9** (Trigonometric Polynomials). *A function $p : \mathbb{R} \to$ is said to be a* trigonometric polynomial *of degree $d$ if there exist complex numbers $\{a_k : k \in \{-d, \ldots, d\}\}$ such that for all $\theta \in \mathbb{R}$,*

$$p(\theta) = \sum_{k=-d}^{d} a_k e^{ik\theta}.$$

▶ **Theorem 10** ([5, Theorem 5.1.2]). *Let $t$ be a degree-$n$ real-valued trigonometric polynomial and $s \in (0, \pi/2]$ be such that $\mu(\{\theta \in [-\pi, \pi) : |t(\theta)| \leq 1\}) \geq 2\pi - s$, where $\mu$ denotes the Lebesgue measure on $\mathbb{R}$. Then, $\sup_{x \in \mathbb{R}} |t(x)| \leq \exp(4ns)$.*

## 3 Lower bounds for maximum phase estimation and Unitary recurrence time

In this section we show lower bounds on the quantum complexity of maximum phase estimation obtained by varying all its parameters (see Section 2.1 and Definition 6). In this section and the next, we refer to the row numbers of Table 1 when stating and proving our bounds.

Recall that for an integer $j \in \{0, 1, \ldots, N-1\}$ and $\delta \in [0, 2\pi)$ we define $M_{j,\delta} = I - (1 - e^{i\delta})|j\rangle\langle j|$. Our lower bounds will be by reduction from the following "Fractional OR with advice" problem, which fits in the framework of the model described in Section 2.1.

▶ **Definition 11** (Fractional OR with advice). *Let $N \geq 2$ be integer, $\delta > 0$. The task $\mathsf{frOR}_{N,\delta,t}$ is:*

- ***Input promise:*** *$U \in \{I, \{M_{j,\delta} : j \in \{1, 2, \ldots, N-1\}\}\}$.*
- ***Advice:*** *When $U = I$ we are given $t$ copies of $|0\rangle$ as advice. When $U = M_{j,\delta}$, we are given $t$ copies of the state $\gamma|j\rangle + \sqrt{1-\gamma^2}|0\rangle$, i.e., part of our starting state is $(\gamma|j\rangle + \sqrt{1-\gamma^2}|0\rangle)^{\otimes t}$.*
- ***Output:*** *With probability at least $2/3$: output 1 if $U = I$, and 0 if $U \neq I$.*

We first show a lower bound on the cost of computing $\mathsf{frOR}_{N,\delta,t}$ when $t = o(1/\gamma^2)$. All of our lower bounds in Table 1 as well as our lower bound for the Unitary recurrence time problem will use this lower bound. We refer the reader to the full version of the paper [25, Appendix A] for the proof. The proof follows along the same lines as Ambainis' adversary lower bound [1, Theorem 4.1] of $\Omega(\sqrt{N})$ queries for the $N$-bit Search problem, but now we additionally take into account the initial advice states and the fact that our input unitaries are only *fractional* versions of phase queries.

▶ **Theorem 12.** *For an integer $N \geq 2$, real numbers $\gamma \geq 1/\sqrt{N}$, $\delta \in [0, \pi]$ and $t = o(1/\gamma^2)$, every algorithm solving $\mathsf{frOR}_{N,\delta,t}$ has cost $\Omega(\sqrt{N}/\delta)$.*

▶ **Lemma 13** (Lower bound for Rows 1,3). *Row 1 (and hence Row 3) has a lower bound of $\Omega(\sqrt{N}/\delta)$.*

**Proof.** A cost-$C$ algorithm $\mathcal{A}$ for $\mathsf{maxQPE}_{N,\delta}$ with $t$ advice states and known eigenbasis of $U$ immediately yields a cost-$C$ algorithm $\mathcal{A}'$ for $\mathsf{frOR}_{N,3\delta,t}$: run $\mathcal{A}$ on the input unitary, output 1 if the output phase is in $[-\delta, \delta]$ modulo $2\pi$, and output 0 otherwise. When $U = I$, the correctness of $\mathcal{A}$ guarantees that with probability at least $2/3$, the value output by $\mathcal{A}$ is in $[-\delta, \delta] \mod 2\pi$. When $U = M_{j,3\delta}$, the correctness of $\mathcal{A}$ guarantees that with probability at least $2/3$, the value output by $\mathcal{A}$ is in $[2\delta, 4\delta]$. For $\delta < 2\pi/5$, we have $[-\delta, \delta] \mod 2\pi \cap [2\delta, 4\delta] \mod 2\pi = \emptyset$. Thus, $\mathcal{A}'$ solves $\mathsf{frOR}_{N,3\delta,t}$ and has cost $C$. Theorem 12 yields the bound $C = \Omega\left(\sqrt{N}/\delta\right)$ when $t = o(1/\gamma^2)$, giving the desired result. ◀

▶ **Lemma 14** (Lower bound for Rows 2,4). *Row 2 (and hence Row 4) has a lower bound of $\Omega(1/\gamma\delta)$.*

**Proof.** We prove the required lower bound for $\mathsf{maxQPE}_{N,\delta}$ with inputs satisfying the promise that $U \in \left\{I_N, \left\{M_{j,3\delta} : j \in \left\{1, 2, \ldots, 1/\gamma^2 - 1\right\}\right\}\right\}$. Because of this assumption, we may take the uniform superposition over the first $1/\gamma^2$ computational basis states as our advice state:

the algorithm should work with such an advice state, since it has overlap $\gamma$ with the top eigenspace for each of the possible $U$. However, an algorithm can prepare such advice states at no cost, so we may assume that the algorithm has no access to advice at all. As in the previous proof, this gives an algorithm of the same cost for $\mathsf{frOR}_{1/\gamma^2,3\delta,0}$ (ignoring all other dimensions). Theorem 12 with $N = 1/\gamma^2$ and $t = 0$ yields the required lower bound of $\Omega(1/\gamma\delta)$. ◀

▶ **Lemma 15** (Lower bound for Rows 5,7). *Row 5 (and hence Row 7) has a lower bound of* $\Omega(\sqrt{N}/\delta)$.

**Proof.** Towards the required lower bound, consider a cost-$C$ algorithm $\mathcal{A}$ solving $\mathsf{maxQPE}_{N,\delta}$ with inputs satisfying the promise $U \in \{I_N, \{M_{j,3\delta} : j \in \{1,2,\ldots,N-1\}\}\}$, and with $t = o(1/\gamma)$ accesses to a unitary that prepares an advice state that has overlap at least $\gamma$ with the target eigenspace. We want to construct an algorithm $\mathcal{A}'$ for $\mathsf{maxQPE}_{N,\delta}$ with the same promised inputs that uses *no* advice, and with cost not much larger than that of $\mathcal{A}$. Note that we may assume $\gamma = o(1)$, since otherwise $t = 0$, so then $\mathcal{A}$ itself already uses no advice.

We first show how an algorithm can itself implement a good-enough advice unitary $A$ quite cheaply. Assuming without loss of generality that $1/3\delta$ is an integer, $U^{1/3\delta}$ is actually a "phase query": if $U = M_{j,3\delta}$, then we have $U^{1/3\delta} = I - 2|j\rangle\langle j|$, which is the diagonal matrix with 1's everywhere except a $-1$ in the $j$th entry; and if $U = I$ then $U^{1/3\delta} = I$. Thus $A$ can start by mapping $|0\rangle$ to a uniform superposition over all indices, and then use Grover's algorithm with $U^{1/3\delta}$ as our query operator to amplify the amplitude of $|j\rangle$ to $\geq \gamma$. We know that $O(\gamma\sqrt{N})$ "Grover iterations" suffice for this (see, for example, [34, Section 7.2] for details). Each Grover iteration would use one phase-query $U^{1/3\delta}$, so the overall cost (number of applications of $U$ and $U^{-1}$) of this advice unitary is $O(\gamma\sqrt{N}/\delta)$. If $U = I$, the state just remains the uniform superposition.

We now have all components to describe $\mathcal{A}'$: Run $\mathcal{A}$, and whenever $\mathcal{A}$ invokes an advice unitary, use the above $A$. Since $\mathcal{A}$ uses at most $t$ advice unitaries, the cost of $\mathcal{A}'$ is at most $C + t \cdot O(\gamma\sqrt{N}/\delta)$. Note that $\mathcal{A}'$ uses no advice at all anymore, and solves $\mathsf{maxQPE}_{N,\delta}$ under the promise that the input unitary satisfies $U \in \{I_N, \{M_{j,3\delta} : j \in \{1,2,\ldots,N-1\}\}\}$. Again, this immediately yields an algorithm of the same cost for $\mathsf{frOR}_{N,3\delta,0}$ as in the previous two proofs. Theorem 12 now implies

$$C + O(t\gamma\sqrt{N}/\delta) = \Omega(\sqrt{N}/\delta),$$

and hence $C = \Omega(\sqrt{N}/\delta)$ since $t = o(1/\gamma)$ ($t \leq c/\gamma$ for sufficiently small constant $c$ also suffices). ◀

▶ **Lemma 16** (Lower bound for Rows 6,8). *Row 6 (and hence Row 8) has a lower bound of* $\Omega\left(1/\gamma\delta\right)$.

**Proof.** Just as in the proof of Lemma 14, we may assume $N = 1/\gamma^2$ by only allowing input unitaries of the form $U \in \left\{I_N, \left\{M_{j,3\delta} : j \in \left\{1,2,\ldots,1/\gamma^2 - 1\right\}\right\}\right\}$. With this assumption, we may assume that we have no access to advice (i.e., $t = 0$) since an algorithm can prepare a good-enough advice state (namely the uniform superposition over all $1/\gamma^2$ basis states) at no cost. This yields the required lower bound of $\Omega(1/\gamma\delta)$ by Lemma 15. ◀

Finally we prove an optimal lower bound for the Unitary recurrence time problem, matching She and Yuen's upper bound (Theorem 8) and resolving one of their open problems [29, Section 2].

**Proof of Theorem 1.** Consider an algorithm $\mathcal{A}$ solving the $(t,\delta)$-recurrence time problem. Restrict to inputs of the form $U \in \{I_N, \{M_{j,3\delta/t} : j \in \{1,2,\ldots,N-1\}\}\}$. When $U = I$ we have $U^t = I$. When $U = M_{j,3\delta/t}$, we have $\|U^t - I\| = |1 - e^{3i\delta}| \geq \delta$ for all $\delta \in [0,1]$. Thus, $\mathcal{A}$ solves $\mathsf{frOR}_{N,3\delta/t,0}$. Theorem 12 yields the required lower bound of $\Omega(t\sqrt{N}/\delta)$. ◀

## 4 Upper bounds for maximum phase estimation

In this section we show upper bounds on the quantum complexity of our 8 variants of maximum phase estimation (see Section 2.1, Definition 6 and Table 1). We require the following generalized maximum-finding procedure, adapted from [2, Lemma 48]; we changed their wording a bit and modified it from minimum-finding to maximum-finding.

▶ **Lemma 17** ([2, Lemma 48]). *There exists a quantum algorithm $\mathcal{M}$ and constant $C > 0$ such that the following holds. Suppose we have a $q$-qubit unitary $V$ such that*

$$V|0\rangle = \sum_{k=0}^{K-1} |\psi_k\rangle|x_k\rangle,$$

*where $x_0 > x_1 > \cdots > x_{K-1}$ are distinct real numbers (written down in finite precision), and the $|\psi_k\rangle$ are unnormalized states. Let $X$ be the random variable obtained if we were to measure the last register, so $\Pr[X = x_k] = \||\psi_k\rangle\|^2$. Let $M \geq C/\sqrt{\Pr[X \geq x_j]}$ for some $j$. Then $\mathcal{M}$ uses at most $M$ applications of $V$ and $V^{-1}$, and $O(qM)$ other gates, and outputs an $x_i \geq x_j$ with probability at least $3/4$ (in particular, if $j = 0$ then $\mathcal{M}$ outputs the maximum).*

▶ Remark 18. It may be verified by going through [2, Lemma 48] that the only applications of $V$ and $V^{-1}$ used by $\mathcal{M}$ are to prepare $V|0\rangle$ starting from $|0\rangle$, and to reflect about $V|0\rangle$.

We can use generalized maximum-finding to approximate the largest eigenphase starting from the ability to prepare a superposition of eigenstates (possibly with some additional workspace qubits):

▶ **Lemma 19.** *There exists a quantum algorithm $\mathcal{B}$ such that the following holds. Suppose we have an $N$-dimensional unitary $U$ with (unknown) eigenstates $|u_0\rangle, \ldots, |u_{N-1}\rangle$ and associated eigenphases $\theta_0, \ldots, \theta_{N-1} \in [0, 2\pi)$. Suppose we also have a unitary $A$ such that*

$$A|0\rangle = \sum_{j=0}^{N-1} \alpha_j |u_j\rangle|\phi_j\rangle,$$

*where $\sum_{j:\theta_j=\theta_{\max}} |\alpha_j|^2 \geq \gamma^2$ and the $|\phi_j\rangle$ are arbitrary (normalized) states. Then $\mathcal{B}$ uses at most $O(1/\gamma)$ applications of $A$ and $A^{-1}$, and $O(\log(1/\gamma)/\gamma\delta)$ applications of $U$ and $U^{-1}$, and with probability at least $2/3$ outputs a number $\theta \in [\theta_{\max} - \delta, \theta_{\max} + \delta] \mod 2\pi$.*

**Proof.** Let $\tilde{V}$ be the unitary that applies phase estimation with unitary $U$, precision $\delta$, and small error probability $\eta$ (to be determined later), on the first register of the state $A|0\rangle$, writing the estimates of the phase in a third register. Then

$$\tilde{V}|0\rangle = \sum_{j=0}^{N-1} \alpha_j |u_j\rangle|\phi_j\rangle|\tilde{\theta}_j\rangle,$$

where, for each $j$, $|\tilde{\theta}_j\rangle$ is a superposition over estimates of $\theta_j$, most of which are $\delta$-close to $\theta_j$.

For the purposes of analysis, we would like to define a "cleaned up" unitary $V$ (very close to $\tilde{V}$) that doesn't have any estimates with error $> \delta$. Let $|\tilde{\theta_j}'\rangle$ be the state obtained from $|\tilde{\theta_j}\rangle$ by removing the estimates that are more than $\delta$-far from $\theta_j$, and renormalizing. Because we ran phase estimation with error probability $\leq \eta$, it is easy to show that $\left\||\tilde{\theta_j}'\rangle - |\tilde{\theta_j}\rangle\right\| = O(\sqrt{\eta})$. Then there exists[6] a unitary $V$ such that $\left\|\tilde{V} - V\right\| = O(\sqrt{\eta})$ and

$$V|0\rangle = \sum_{j=0}^{N-1} \alpha_j |u_j\rangle|\phi_j\rangle|\tilde{\theta_j}'\rangle = \sum_{k=0}^{K-1} |\psi_k\rangle|x_k\rangle,$$

where the $x_k$ are the distinct estimates that have support in the last register, and the $|\psi_k\rangle$ are (unnormalized) superpositions of the $|u_j\rangle|\phi_j\rangle$'s that are associated with those estimates.

The largest $x_k$'s are good estimates of $\theta_{\max}$. Algorithm $\mathcal{B}$ now applies the maximum-finding algorithm $\mathcal{M}$ of Lemma 17 with the unitary $\tilde{V}$. Let us first analyze what would happen if $\mathcal{B}$ used the cleaned-up $V$ instead of $\tilde{V}$. Let $X$ denote the random variable obtained if we measure the last register, and note that $\Pr[X \geq \theta_{\max} - \delta] \geq \sum_{j:\theta_j=\theta_{\max}} |\alpha_j|^2 \geq \gamma^2$ because all estimates in $V|0\rangle$ have error $\leq \delta$. Hence $\mathcal{B}$ would use $O(1/\gamma)$ applications of $V$ and $V^{-1}$ to find a $\theta \in [\theta_{\max} - \delta, \theta_{\max} + \delta]$ with success probability $\geq 3/4$. Algorithm $\mathcal{B}$ will actually use $\tilde{V}$ and $\tilde{V}^{-1}$ instead of $V$ and $V^{-1}$, which (because errors in quantum circuits add at most linearly) incurs an overall error in operator norm of $\leq O(\sqrt{\eta}) \cdot O(1/\gamma)$. Choosing $\eta \ll \gamma^2$, this overall error can be made an arbitrarily small constant. The success probability of the algorithm can drop slightly below $3/4$ now, but is still $\geq 2/3$.

It remains to analyze the cost of $\mathcal{B}$. Each $\tilde{V}$ uses 1 application of $A$, and $O(\log(1/\eta)/\delta) = O(\log(1/\gamma)/\delta)$ applications of $U$ and $U^{-1}$ for phase estimation (Theorem 2), so $\mathcal{B}$ uses $O(1/\gamma)$ applications of $A$ and $A^{-1}$, and $O(\log(1/\gamma)/\gamma\delta)$ applications of $U$ and $U^{-1}$ in total. ◀

The upper bounds for our 8 variants of phase estimation (see Table 1) will all follow from this. We start with the 4 odd-numbered rows, where it turns out the advice is not actually needed to meet our earlier lower bounds. The next proof is basically the same as [2, Lemma 50] about estimating the minimal eigenvalue of a Hamiltonian (this improved slightly upon [27]; see also [14, Lemma 3.A.4]).

▶ **Lemma 20** (Upper bound for Rows 1, 3, 5, 7)**.** *There is an algorithm that uses no advice and solves the case in Row 3 (and hence in Rows 1, 5, and 7 as well) with cost* $O(\sqrt{N}\log(N)/\delta)$.

**Proof.** Let $A$ be the unitary that maps $|0\rangle$ to the maximally entangled state in $N$ dimensions. This state can be written in any orthonormal basis, including the (unknown) eigenbasis of $U$:

$$A|0\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle|j\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |u_j\rangle|\overline{u_j}\rangle,$$

where $|\overline{u_j}\rangle$ denotes the entry-wise conjugated version of $|u_j\rangle$. Applying Lemma 19 with this $A$, $|\phi_j\rangle = |\overline{u_j}\rangle$, and $\gamma = 1/\sqrt{N}$ gives the result. ◀

The next two lemmas cover the 4 cases where the advice states/unitaries *are* helpful.

▶ **Lemma 21** (Upper bound for Rows 6, 8)**.** *There is a quantum algorithm that uses* $O(1/\gamma)$ *applications of the advice unitary (and its inverse) and solves the case in Row 8 (and hence the case in Row 6 as well) with cost* $O(\log(1/\gamma)/\gamma\delta)$.

[6] This is fairly easy to show, see e.g. [9, proof of Theorem 2.4 in Appendix A].

**Proof.** Apply Lemma 19 with the unitary $A$ that maps $|0\rangle$ to $|\alpha\rangle$, with empty states $|\phi_j\rangle$. ◀

▶ **Lemma 22** (Upper bound for Rows 2, 4)**.** *There is a quantum algorithm that uses $O(1/\gamma^2)$ copies of the advice state and solves the case in Row 4 (and hence in Row 2) with cost $O(\log(1/\gamma)/\gamma\delta)$.*

**Proof.** We will build upon the algorithm for Row 8 of Lemma 21. By Remark 18 and the algorithm in Lemma 21, its $O(1/\gamma)$ applications of the advice unitary $A$ and its inverse $A^{-1}$ are only used there for two purposes: (1) to prepare a copy of the advice state $A|0\rangle = |\alpha\rangle$, and (2) to reflect about $|\alpha\rangle$. We now want to replace these applications of $A$ by using copies of the advice state. For (1) this is obvious. Assume the algorithm for Row 8 uses (2) at most $C/\gamma$ times, for some constant $C$. To implement these reflections, we will invoke the result of Lloyd, Mohseni, and Rebentrost [24] (see also [18]), who showed that given a number $t > 0$ and $O(t^2/\eta)$ copies of a mixed quantum state $\rho$, one can implement the unitary $e^{it\rho}$ up to error $\eta$ (in diamond-norm difference between the intended unitary and the actually-implemented channel). We will use this result with $\rho = |\alpha\rangle\langle\alpha|$, $t = \pi$, $\eta = \gamma/(100C)$, noting that the implemented unitary $e^{i\pi|\alpha\rangle\langle\alpha|} = I - 2|\alpha\rangle\langle\alpha|$ is a reflection about $|\alpha\rangle$ (up to a global minus sign that doesn't matter).

Accordingly, we can implement the $\leq C/\gamma$ reflections used by the algorithm for Row 8 using $O(1/\gamma^2)$ copies of $|\alpha\rangle$, each reflection implemented with error $\leq \eta$. Because errors in quantum circuits add at most linearly, the overall error between the algorithm of Row 8 and our simulation of it (using copies of $|\alpha\rangle$) is at most $\eta \cdot C/\gamma \leq 1/100$. Hence we obtain an algorithm for Row 4 that uses $O(1/\gamma^2)$ copies of $|\alpha\rangle$ and has the same cost $O(\log(1/\gamma)/\gamma\delta)$ as the algorithm of Row 8. ◀

## 5 Tight bounds for phase estimation with small error probability

Here we prove our lower bound for quantum algorithms solving phase estimation with precision $\delta$ and error probability at most $\varepsilon$, Theorem 3, which follows from Claims 23 and 24 below.

▷ Claim 23. For all integers $N \geq 2$ and all $\varepsilon, \delta \in (0, 1/2)$, if there is a cost-$d$ algorithm solving $\mathsf{QPE}_{N,\delta,\varepsilon}$, then there is a cost-$d$ algorithm solving $\mathsf{dist}_{N,\delta,\varepsilon}$.

Proof. Consider an algorithm $\mathcal{A}$ of cost $d$ that solves $\mathsf{QPE}_{N,\delta,\varepsilon}$. We construct below an algorithm $\mathcal{A}'$ of cost $d$ solving $\mathsf{dist}_{N,\delta,\varepsilon}$. Let $U \in U(N)$ be the input. The following is the description of $\mathcal{A}'$:

1. Run $\mathcal{A}$ with inputs $U$ and $|0\rangle$.
2. Output 1 if the output of $\mathcal{A}$ is in $[-\delta, \delta] \mod 2\pi$, and output 0 otherwise.

Clearly $\mathcal{A}'$ is a valid algorithm, as far as access to input and allowed operations are concerned, since its initial state is $|0\rangle$, it applies $U, U^{-1}$, some unitaries independent of $U$, and finally performs a two-outcome projective measurement to determine the output bit. The cost of $\mathcal{A}'$ is $d$.

The correctness follows along the same lines as the proofs in Section 3. We prove correctness here for completeness. First note that $|0\rangle$ is an eigenstate of all $U \in \{I\} \cup \{U_\theta : \theta \notin [-3\delta, 3\delta] \mod 2\pi\}$. When $U = I$, the correctness of $\mathcal{A}$ guarantees that with probability at least $1 - \varepsilon$, the value output by $\mathcal{A}$ is in $[-\delta, \delta] \mod 2\pi$. When $U = U_\theta$, the correctness of $\mathcal{A}$ guarantees that with probability at least $1 - \varepsilon$, the value output by $\mathcal{A}$ is in $[\theta - \delta, \theta + \delta] \mod 2\pi$. For $\theta \notin [-3\delta, 3\delta] \mod 2\pi$ we have $[-\delta, \delta] \mod 2\pi \cap [\theta - \delta, \theta + \delta] \mod 2\pi = \emptyset$ since $\delta < 1/2 < 2\pi/5$, and hence $\mathcal{A}'$ solves $\mathsf{dist}_{N,\delta,\varepsilon}$. ◁

We next show a lower bound for the cost of algorithms computing $\mathsf{dist}_{N,\delta,\varepsilon}$.

▷ Claim 24. For all integers $N \geq 2$, $\varepsilon, \delta \in (0, 1/2)$, every algorithm for $\mathsf{dist}_{N,\delta,\varepsilon}$ has cost $\Omega\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)$.

In order to prove Claim 24, we first show that amplitudes of basis states in low-cost algorithms that run on $U_\theta$ are low-degree trigonometric polynomials in $\theta$. This is analogous to the fact that amplitudes of basis states in query algorithms for Boolean functions are low-degree (algebraic) polynomials in the input variables [3, Lemma 4.1], and our proof is inspired by theirs.

▷ Claim 25. Let $t > 0$ be a positive integer and let $\theta \in [0, 2\pi]$. Consider a quantum circuit that has starting state $|0\rangle$, uses an arbitrary number of $\theta$-independent unitaries, uses $t$ applications of controlled-$U_\theta$ and controlled-$U_\theta^{-1}$ in total, and performs no intermediate measurements. Then the amplitudes of basis states before the final measurement are degree-$t$ trigonometric polynomials in $\theta$.

Proof. We prove this by induction on $t$. The claim is clearly true when $t = 0$ since all amplitudes are constants in this case. For the inductive step, suppose the claim is true for $t = d$. Let $|\psi_d\rangle$ denote the state of the circuit just before the application of the $(d+1)$th application of $U_\theta$ (the argument for $U_\theta^{-1}$ is similar, and we skip it). By the inductive hypothesis, we have

$$|\psi_d\rangle = \sum_w \sum_{b\in\{0,1\}} \sum_{j=0}^{N-1} p_{j,b,w}(\theta)|j\rangle|b\rangle|w\rangle,$$

where the first register is where $U_\theta$ and $U_\theta^{-1}$ act, the second register is the control qubit, and the last register represents the workspace (i.e., $U_\theta$ and $U_\theta^{-1}$ do not act on this register), and each $p_{j,b,w}$ is a trigonometric polynomial of degree at most $d$ in $\theta$. For a basis state $|j\rangle|b\rangle|w\rangle$, we have

$$U_\theta|j\rangle|b\rangle|w\rangle = \begin{cases} e^{i\theta}|0\rangle|b\rangle|w\rangle & \text{if } j = 0 \text{ and } b = 1 \\ |j\rangle|b\rangle|w\rangle & \text{otherwise.} \end{cases}$$

In both cases, the amplitudes of the basis states after the application of $U_\theta$ are degree-$(d+1)$ trigonometric polynomials in $\theta$. After the last application of $U_\theta$ the algorithm will apply an input-independent unitary. The amplitudes after that unitary are linear combinations of the amplitudes before, which won't increase degree. This concludes the inductive step, and hence the theorem. ◁

**Proof of Claim 24.** Consider a cost-$t$ algorithm $\mathcal{A}'$ solving $\mathsf{dist}_{N,\delta,\varepsilon}$. Claim 25 implies that on input $U_\theta$, the amplitudes of the basis states before the final measurement are degree-$t$ trigonometric polynomials in $\theta$. The acceptance-probability polynomial $p : \mathbb{R} \to \mathbb{R}$ given by $p(\theta) := \Pr[\mathcal{A}'(U_\theta) = 1]$ is a degree-$2t$ trigonometric polynomial, because it is the sum of squares of moduli of certain amplitudes, and each of these squares is a degree-$2t$ trigonometric polynomial. The correctness of the algorithm ensures that $p(0) \in [1-\varepsilon, 1]$ and $p(\theta) \in [0, \varepsilon]$ for all $\theta \notin [-3\delta, 3\delta] \mod 2\pi$. See Figure 1 for a visual depiction of the behaviour of $p$ for $\theta \in [-\pi, \pi)$.

Scaling by a global factor of $1/\varepsilon$, we obtain a trigonometric polynomial $q$ of degree $2t$ satisfying:

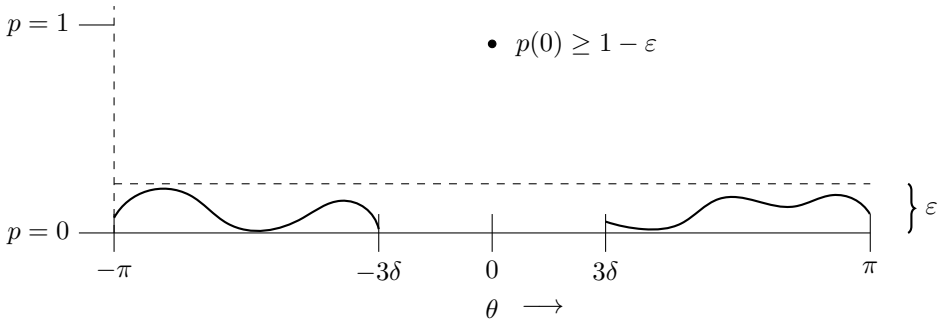



**Figure 1** Acceptance probability $p$ of $\mathcal{A}'$ as a function of $\theta$ in the proof of Claim 24.

- $q(0) \geq (1-\varepsilon)/\varepsilon > 1/(2\varepsilon)$, and
- $q(\theta) \in [0,1]$ for all $\theta \in [-\pi, \pi) \setminus [-3\delta, 3\delta]$.

Thus, Theorem 10 is applicable with $s = 6\delta$, which implies $1/(2\varepsilon) \leq \sup_{x\in\mathbb{R}} |q(x)| \leq \exp(24t\delta)$. By taking logarithms and rearranging we get $t = \Omega\left(\frac{1}{\delta}\log\frac{1}{\varepsilon}\right)$, proving the theorem. ◀

## 6 Conclusion

In this paper we considered several natural variants of the fundamental phase estimation problem in quantum computing, and proved essentially tight bounds on their cost in each setting. As an immediate application of one of our bounds, we resolved an open question of [29, Section 2].

We mention two interesting questions in the first variant of phase estimation we considered, where an algorithm is given a number of copies of advice states/unitaries instead of black-box access to a perfect eigenstate as in the basic phase estimation setup. First, are the logarithmic overheads in the input dimension $N$ and the inverse of the overlap $\gamma$ in our upper bounds (see Table 1) necessary, or can we give tighter upper bounds? Second, can we show the $\log(1/\varepsilon)$-dependence on the error probability also in the advice-guided case, like we did for basic phase estimation (Theorem 3)?

## References


1 Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. Earlier version in STOC'00. doi:10.1006/jcss.2002.1826.

2 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-solvers: Better upper and lower bounds. *Quantum*, 4:230, 2020. arXiv:1705.01843. Earlier version in FOCS'17. doi:10.22331/q-2020-02-14-230.

3 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. arXiv:quant-ph/9802049. Earlier version in FOCS'98. doi:10.1145/502090.502097.

4 Arvid J. Bessen. Lower bound for quantum phase estimation. *Physical Review A*, 71(4):042313, 2005.

5 Peter Borwein and Tamás Erdélyi. *Polynomials and polynomial inequalities*, volume 161. Springer Science & Business Media, 1995.

6 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74. AMS, 2002. arXiv:quant-ph/0005055.

7 Harry Buhrman, Richard Cleve, Ronald de Wolf, and Christof Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of 40th IEEE FOCS*, pages 358–368, 1999. arXiv:cs/9904019. doi:10.1109/SFFCS.1999.814607.

8 Chris Cade, Marten Folkertsma, and Jordi Weggemans. Complexity of the guided local Hamiltonian problem: Improved parameters and extension to excited states, 2022. arXiv:2207.10097.

9 Yanlin Chen and Ronald de Wolf. Quantum algorithms and lower bounds for linear regression with norm constraints. In *Proceedings of 50th ICALP*, 2023. To appear. arXiv:2110.13086.

10 Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. In *Proceedings of the Royal Society of London*, volume A454, pages 339–354, 1998. arXiv:quant-ph/9708016.

11 Yimin Ge, Jordi Tura, and J. Ignacio Cirac. Faster ground state preparation and high-precision ground energy estimation with fewer qubits. *Journal of Mathematical Physics*, 60(2):022202, 2019. arXiv:1712.03193.

12 Sevag Gharibian, Ryu Hayakawa, François Le Gall, and Tomoyuki Morimae. Improved hardness results for the guided local Hamiltonian problem, 2022. doi:10.48550/arXiv.2207.10250.

13 Sevag Gharibian and Francçois Le Gall. Dequantizing the quantum singular value transformation: hardness and applications to quantum chemistry and the quantum PCP conjecture. In *Proceedings of 54th ACM STOC*, pages 19–32, 2022. arXiv:2111.09079.

14 András Gilyén. *Quantum Singular Value Transformation & Its Algorithmic Applications*. PhD thesis, University of Amsterdam, 2019.

15 András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of 51st ACM STOC*, pages 193–204, 2019. arXiv:1806.01838.

16 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996. arXiv:quant-ph/9605043.

17 Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for solving linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009. arXiv:0811.3171.

18 Shelby Kimmel, Cedric Yen-Yu Lin, Guang Hao Low, Maris Ozols, and Theodore J. Yoder. Hamiltonian simulation with optimal sample complexity. *npj Quantum Information*, 3(13), 2017. arXiv:1608.00281.

19 A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995. arXiv:quant-ph/9511026.

20 Yao-Ting Lin. A note on quantum phase estimation. arXiv:2304.02241, 2023.

21 Lin Lin and Yu Tong. Near-optimal ground state preparation. *Quantum*, 4(372), 2020. arXiv:2002.12508. doi:10.22331/q-2020-12-14-372.

22 Lin Lin and Yu Tong. Heisenberg-limited ground state energy estimation for early fault-tolerant quantum computers. *PRX Quantum*, 3(010318), 2022. arXiv:2102.11340.

23 Noah Linden and Ronald de Wolf. Average-case verification of the quantum Fourier transform enables worst-case phase estimation. *Quantum*, 6(872), 2022. arXiv:2109.1021. doi:10.22331/q-2022-12-07-872.

24 Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10:631–633, 2013. arXiv:1307.0401.

25 Nikhil S. Mande and Ronald de Wolf. Tight bounds for quantum phase estimation and related problems. *CoRR*, 2023. doi:10.48550/arXiv.2305.04908.

26 Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of 31st ACM STOC*, pages 384–393, 1999. doi:10.1145/301250.301349.

27 David Poulin and Pawel Wocjan. Sampling from the thermal quantum Gibbs state and evaluating partition functions with a quantum computer. *Physical Review Letters*, 103(22):220502, 2009. arXiv:0905.2199. doi:10.1103/PhysRevLett.103.220502.

28 Patrick Rall. Faster coherent quantum algorithms for phase, energy, and amplitude estimation. *Quantum*, 5(566), 2021. arXiv:2103.09717. doi:10.22331/q-2021-10-19-566.

**29** Adrian She and Henry Yuen. Unitary property testing lower bounds by polynomials. In *14th Innovations in Theoretical Computer Science Conference, ITCS*, volume 251 of *LIPIcs*, pages 96:1–96:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ITCS.2023.96.

**30** Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Earlier version in FOCS'94. arXiv:quant-ph/9508027.

**31** Kianna Wan, Mario Berta, and Earl T. Campbell. A randomized quantum algorithm for statistical phase estimation. *Physical Review Letters*, 129(030503), 2022. arXiv:2110.12071.

**32** Jordi Weggemans, Marten Folkertsma, and Chris Cade. Guidable local Hamiltonian problems with implications to heuristic Ansätze state preparation and the quantum PCP conjecture, 2023. arXiv:2302.11578.

**33** Ronald de Wolf. A note on quantum algorithms and the minimal degree of $\varepsilon$-error polynomials for symmetric functions. *Quantum Information and Computation*, 8(10):943–950, 2008. arXiv:0802.1816.

**34** Ronald de Wolf. Quantum computing: Lecture notes, 2019. arXiv:1907.09415, version 5. arXiv:1907.09415.

# A Fine-Grained Classification of the Complexity of Evaluating the Tutte Polynomial on Integer Points Parameterized by Treewidth and Cutwidth

**Isja Mannens**
Utrecht University, The Netherlands

**Jesper Nederlof**
Utrecht University, The Netherlands

## Abstract

We give a fine-grained classification of evaluating the Tutte polynomial $T(G; x, y)$ on all integer points on graphs with small treewidth and cutwidth. Specifically, we show for any point $(x, y) \in \mathbb{Z}^2$ that either

- $T(G; x, y)$ can be computed in polynomial time,
- $T(G; x, y)$ can be computed in $2^{O(tw)}n^{O(1)}$ time, but not in $2^{o(ctw)}n^{O(1)}$ time assuming the Exponential Time Hypothesis (ETH),
- $T(G; x, y)$ can be computed in $2^{O(tw \log tw)}n^{O(1)}$ time, but not in $2^{o(ctw \log ctw)}n^{O(1)}$ time assuming the ETH,

where we assume tree decompositions of treewidth $tw$ and cutwidth decompositions of cutwidth $ctw$ are given as input along with the input graph on $n$ vertices and point $(x, y)$.

To obtain these results, we refine the existing reductions that were instrumental for the seminal dichotomy by Jaeger, Welsh and Vertigan [Math. Proc. Cambridge Philos. Soc'90]. One of our technical contributions is a new rank bound of a matrix that indicates whether the union of two forests is a forest itself, which we use to show that the number of forests of a graph can be counted in $2^{O(tw)}n^{O(1)}$ time.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** Width Parameters, Parameterized Complexity, Tutte Polynomial

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.82

**Related Version** *Full Version*: `https://arxiv.org/abs/2307.01046`

**Supplementary Material** *Software (MATLAB Code)*: `https://github.com/isja-m/ForestRank4-5` archived at `swh:1:dir:2e6936582c19e5fd2f127b3d1e601ecb9a1136f1`

**Funding** Both authors are supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 853234).

## 1 Introduction

We study the parameterized complexity of computing the Tutte Polynomial. The Tutte polynomial is a graph invariant that generalizes any graph invariant that satisfies a linear deletion-contraction recursion. Such invariants include the chromatic, flow and Jones polynomials, as well as invariants that count structures such as the number of forests or the number of spanning subgraphs. Due to its generality the Tutte polynomial is of great interest to a variety of fields, including knot theory, statistical physics and combinatorics.




For a number of these fields it is important to understand how difficult it is to compute the Tutte polynomial. A series of papers, culminating in the work by Jaeger, Vertigan, and Welsh [15] has given a complete dichotomy showing that the problem of evaluating the Tutte polynmial is #P-hard on all points except on the following *special points* on which it is known to be computable in polynomial time:

$$(1,1),(-1,-1),(0,-1),(-1,0),(i,-i),(-i,i),(j,j^2),(j^2,j),\ H_1 \tag{1}$$

where $j = e^{2\pi i/3}$ and $i = \sqrt{-1}$, and $H_\alpha$ denotes the hyperbola $\{(x,y) : (x-1)(y-1) = \alpha\}$. These hyperbolic curves turn out to be of great importance to understanding the complexity of the Tutte Polynomial, as the problem is generally equally hard on all points of the same curve, except for the *special points* listed in (1).

Further refinements of the result by [15] have since been made: Among others, a more fine-grained examination of the complexity was done by Brand et al. [5] (building on earlier work by Dell et. al. [12]): they showed that for almost all points the Tutte polynomial cannot be evaluated in $2^{o(n)}$ time on $n$-vertex graphs, assuming (a weaker counting version of) the Exponential Time Hypothesis. This is tight because, on the positive side, Björklund et al. [2] showed that the Tutte polynomial can be evaluated on any point in $2^n n^{O(1)}$ time.

Another perspective worth examining is that of the parameterized complexity of the problem, when parameterized by *width measures*. This is a rapidly evolving field within parameterized complexity.[1] Intuitively, it is concerned with the effects of structural properties of the given input graph on its complexity. This often generates results that have greater practical value and give a deeper understanding of the problem, in comparison with classical worst-case analysis. It is therefore natural to ask what a complexity classification for the Tutte Polynomial would look like in this parameterized context.

For the specific subject of evaluating the Tutte polynomial parameterized by width measures, research has already been done in this area over twenty years ago: Noble [21] has given a polynomial time algorithm for evaluation the Tutte Polynomial on bounded treewidth graphs. Noble mostly focused on the dependence on the number of vertices and edges, and showed each point of the Tutte polynomial can be evaluated in linear time, assuming the treewidth of the graph is constant. See also an independently discovered (but slower) algorithm by Andrzejak [1]. However, this glances over the *exponential* part of the runtime, i.e. the dependence on the treewidth. Since this is typically the bottleneck, recent work aims to refine our understanding of this exponential dependence with upper and lower bounds on complexity of the problem in terms of this parameter that match in a fine-grained sense.

In this work, we extend this research line and determine the fine-grained complexity for each integer point $(x,y)$ of the problem of evaluating the Tutte polynomial $(x,y)$. As was done in previous works, we base our lower bounds on the Exponential Time Hypothesis (ETH) and the Strong Exponential Time Hypothesis (SETH) formulated by Impagliazzo and Paturi [14]. For a given width parameter $k$, the former will be used to exclude run times of the form $k^{o(k)}n^{O(1)}$, while the latter will be used to exclude run times of the form $(c-\epsilon)^k n^{O(1)}$ for some constant $c$ and any $\epsilon > 0$.

Specifically we consider the *treewidth*, *pathwidth* and *cutwidth* of the graph. The first two, in some sense, measure how close the graph is to looking like a tree or path respectively. The cutwidth measures how many edges are layered on top of each other when the vertices are placed in any linear order. We will more precisely define these parameters in the preliminaries.

[1] For example, the biennial Workshop on Graph Classes, Optimization, and Width Parameters (GROW) already had its 10'th edition recently `https://conferences.famnit.upr.si/event/22/`.

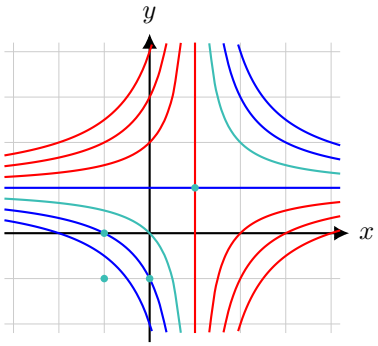



**Figure 1** The red points have time complexity of the form $k^{O(k)}$, the blue points have time complexity of the form $O(c^k)$ for some constant $c$ and the green points have polynomial time complexity.

Width measures in particular are interesting because instances where such structural parameters are small come up a lot in practice. For example, the curve $H_2$ corresponds to the partition function of the Ising model, which is widely studied in statistical physics, on graphs with particular topology such as lattice graphs or open/closed Cayley trees ([18]). In all such graphs with $n$ vertices, even the cutwidth (the largest parameter we study) is at most $O(\sqrt{n})$.

## 1.1 Our contributions

Our classification handles points $(x, y)$ differently based on whether $(x-1)(y-1)$ is negative, zero or positive, and reads as follows:

▶ **Theorem 1.1.** *Let $G$ be a graph with given tree, path and cut decompositions of width* tw, pw *and* ctw *respectively. Let $(x, y) \in \mathbb{Z}^2$ be a non-special point, then up to some polynomial factor in $|V(G)|$, the following holds.*

1. *If $(x-1)(y-1) < 0$ or $x = 1$, then $T(G; x, y)$ can be computed in time $\mathrm{tw}^{O(\mathrm{tw})}$ and cannot be computed in time $\mathrm{ctw}^{o(\mathrm{ctw})}$ under ETH.*
2. *If $y = 1$, then $T(G; x, y)$ can be computed in time $O(4^{\mathrm{pw}})$ or $O(64^{\mathrm{tw}})$ and cannot be computed in time $2^{o(\mathrm{ctw})}$ under ETH.*
3. *If $(x-1)(y-1) = q > 1$, then $T(G; x, y)$ can be computed in time $O(q^{\mathrm{tw}})$. Furthermore,*
   a. *if $x \neq 0$, then $T(G; x, y)$ cannot be computed in time $O((q-\epsilon)^{\mathrm{ctw}})$ under SETH.*
   b. *if $x = 0$, then $T(G; x, y)$ cannot be computed in time $O((q-\epsilon)^{\mathrm{pw}})$ and $O(q-\epsilon)^{\mathrm{ctw}/2})$ under SETH.*

This is a fine-grained classification for evaluating the Tutte polynomial at any given integer point, simultaneously for all the parameters treewidth, pathwidth and cutwidth. This is because if a graph has cutwidth ctw, pathwidth pw and treewidth tw, then $\mathrm{tw} \leq \mathrm{pw} \leq \mathrm{ctw}$. Our result implies that, for evaluating the Tutte polynomial at a given integer point, it does not give a substantial advantage to have small cutwidth instead of small treewidth. This is somewhat surprising since, for example, for computing the closely related chromatic number of a graph there exists a $2^{\mathrm{ctw}} n^{O(1)}$ time algorithm, but any $\mathrm{pw}^{o(\mathrm{pw})}\, n^{O(1)}$ time algorithm would contradict the ETH [19].

Of particular interest are the upper bounds in Case 2. for the points $\{(x, y) : y = 1\}$, which are closely related to the problem of computing the number of forests in the input graph. One reason why this results stands out in particular is that it indicates an inherent

asymmetry between the $x$- and $y$-axes, in this parameterized setting. In the general setting, problems related to the Tutte Polynomial often have a natural dual problem, which one can obtain by interchanging the $x$- and $y$-coordinates. For example the chromatic polynomial can be found (up to some computable term $f$) as $\chi_G(\lambda) = f(\lambda)T(1-\lambda, 0)$, while the flow polynomial can be found as $C_G(\lambda) = g(\lambda)T(0, 1-\lambda)$. These two problems are equivalent on planar graphs, in the sense that the chromatic number of a planar graph is equal to the flow number of its dual graph.

We note that for this curve we have an ETH bound, while for the other results of the form $c^{\mathrm{tw}}n^{O(1)}$ we have a stronger SETH bound. We suspect that a $(4-\epsilon)^{\mathrm{ctw}}n^{O(1)}$ lower bound for any $\epsilon > 0$, based on SETH, also holds for evaluating $T(G; 2, 1)$, but that it will take significant additional technical effort.

**Techniques.** In order to get the classification, our first step follows the method of [15] to reduce the evaluation of $T(G; x, y)$ for all points in hyperbola $H_\alpha = \{(x, y) : (x-1)(y-1) = \alpha\}$ to the evaluation to a *single* point in $H_\alpha$. This is achieved in [15] by some graph operations (*stretch* and *thickening*), but these may increase the involved width parameters. We refine these operations in Section 3 to avoid this.

With this step being made, several cases of Theorem 1.1 then follow from a combination of new short separate and non-trivial arguments and previous work (including some very recent work such as [8, 13]).

However, for the upper bound in Case **2.** of Theorem 1.1, our proof is more involved. To get our upper bound, we introduced the *forest compatibility matrix*. Its rows and columns are indexed with forests (encoded as partitions indicating their connected components). An entry in this matrix indicates whether the union of the two forests forms a forest itself. This matrix is closely related to matrices playing a crucial role in the Cut and Count method [11] and rank based method [4] to quickly solve connectivity problems on graphs with small tree-width. However, the previous rank upper bounds do not work for bounding the rank of the forest compatibility matrix over the reals since we check for *acyclicity* instead of *connectivity*. We nevertheless show that this the rank of this matrix is $4^n$; in fact the set of non-crossing partitions forms a basis of this matrix. We prove this via an inductive argument that is somewhat similar to the rank bound of $2^{n/2-1}$ of the matchings connectivity matrix over $GF(2)$ from [10]. Subsequently, we show how to use this insight to get a $2^{O(\mathrm{tw})}$ algorithm to evaluate $T(G; 1, 2)$ (i.e. counting the number of spanning forests).

## 1.2 Organization

The remainder of this paper supports Theorem 1.1, although some slightly less interesting cases (being the upper bound in Case **1.**) are deferred to the full version of the paper [20]. Proofs of Lemmas and Theorems indicated with † are also deferred to the full version [20].

In Section 2 we describe some preliminaries. In Section 3 we show how to reduce the task of computing all points along a hyperbola curve to a single point. We now describe where each part of Theorem 1.1 can be found in the paper. The lower bound in Case **1.** is given in Theorem 5.7 and 5.3. The lower bound in Case **2.** is by Dell et al. [12]. The upper bound in Case **2.** is given in Section 4 (specifically, Theorems 4.17 and 4.18). The lower bound in Case **3.** is given in Theorem 5.1 (for $q = 2$) and Theorem 5.4 (for $q > 2$). The upper bound bound in **3.** is given in Theorem 5.2 (for $q = 2$) and Theorem 5.5 (for $q > 2$).

## 2 Preliminaries

**Computational Model.** In this paper we frequently have real (and some inteormediate lemma's are even stated for even complex) numbers as intermediate results of computations. However, as is common in this area we work in the word RAM model in which all basic arithmetic operations with such numbers can be done in constant time, and therefore this does not influence our running time bounds.

**Interpolation.** Throughout this paper we will use interpolation to derive a polynomial, given a finite set of evaluations of said polynomial. For our purposes it suffices to note that this can be done in polynomial time, for example by solving the system of linear equations given by the Vandermonde matrix and the evaluations (see e.g. [7, Section 30.1]).

▶ **Lemma 2.1.** *Given pairs $(x_0, y_0), \ldots, (x_d, y_d)$, there exists an algorithm which computes the unique degree $d$ polynomial $p$ such that $p(x_i) = y_i$ for $i = 0, \ldots, d$ and runs in time $O(d^3)$.*

### 2.1 The Tutte polynomial

There are multiple ways of defining the Tutte polynomial. In this paper we will only need the following definition

$$T(G; x, y) = \sum_{A \subseteq E} (x-1)^{k(A)-k(E)}(y-1)^{k(A)+|A|-|V|},$$

where $k(A)$ denotes the number of connected components of the graph $(V, A)$. We will often use the following notation

$$H_\alpha = \{(x, y) : (x-1)(y-1) = \alpha\}.$$

Note that these curves form hyperbolas and that for $\alpha = 0$ the hyperbola collapses into two orthogonal, straight lines. We refer to these two lines as separate curves

$$H_0^x = \{(x, y) : x = 1\},$$
$$H_0^y = \{(x, y) : y = 1\}.$$

Throughout the paper we will refer to the problem of finding the value of $T(G; a, b)$ for an individual point as *computing the Tutte polynomial on* $(a, b)$. We will often restrict the Tutte polynomial to a one-dimensional curve $H_\alpha$. Note that in this case the polynomial can be expressed as a univariate polynomial[2]

$$T_\alpha(G; t) := T\left(G; \frac{\alpha}{t} + 1, t + 1\right).$$

We refer to the problem of finding the coefficients of $T_\alpha$ as *computing the Tutte polynomial along* $H_\alpha$.

As mentioned in the introduction, the Tutte polynomial is known to be computable in polynomial time on the points

$$(1,1), (-1,-1), (0,-1), (-1,0), (i,-i), (-i,i), (j,j^2), (j^2,j) \tag{2}$$

and along the curve $H_1$ and it is $\#P$ to evaluate it on any other point. We call the points listed in (2), along with the points on the curve $H_1$ *special points*. See [15] for more details.

[2] Note that one can get rid of the negative powers of $t$ in the following expression, by multiplying the whole polynomial by some power of $t$.

## 2.2 Width measures

We consider the width measures *treewidth*, *pathwidth* and *cutwidth* of a graph $G$ (denoted respectively with $tw(G)$, $pw(G)$ and $ctw(G)$). We use standard notation (such as introduced in [9]); see the appendix

## 2.3 Brylawski's tensor product formula

In Section 3 we will make use of Brylawski's tensor product formula [6] to reduce the computation of $T(G; x', y')$ to that of $T(G'; x, y)$ for some other point $(x', y')$ and some other graph $G'$. The original formula is formulated in terms of pointed matroids, however we will only need the formulation for (multi)graphs. Before we can state the formula, we first need to introduce some notation.

Given graphs $G$ and $H$, where an edge $e \in E(H)$ is labeled as a special edge, we define the *pointed tensor product*[3] $G \otimes_e H$ of $G$ and $H$ as the graph given by the following procedure. For every edge $f \in E(G)$ we first create a copy $H_f$ of $H$, then identify $f$ with the copy of the edge $e$ in $H_f$ and finally remove the edge f (and thus also the edge $e$) from the graph.

Intuitively it might be easier to think of this product as replacing every edge of $G$ with a copy $H \setminus e$, where two of the vertices in $H$ are designated as gluing points. For example one could replace every edge with a path of length $k$ by taking as $H$ the cycle $C_{k+1}$ on $k+1$ vertices, as seen in figure 2.

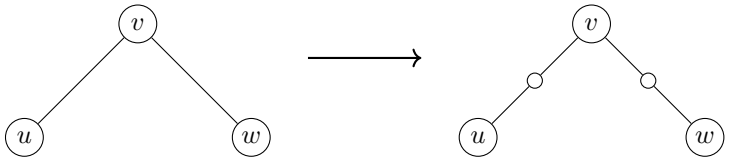



**Figure 2** The pointed tensor product of the left-hand graph with a 3-cycle is given by the right-hand graph.

Note that this is not always well-defined, as one can choose which endpoint is identified with which. It turns out that this choice does not affect the graphic matroid of $G \otimes_e H$ and thus it does not affect the resulting Tutte polynomial. In this paper we will only consider graphs $H$ that are symmetric over $e$ and thus the product is actually well-defined.

We are now ready to state Brylawski's tensor product formula. Let $T_C$ and $T_L$ be the unique polynomials that satisfy the following system of equations

$$(x-1)T_C(H; x, y) + T_L(H; x, y) = T(H \setminus e; x, y)$$
$$T_C(H; x, y) + (y-1)T_L(H; x, y) = T(H/e; x, y).$$

We define

$$x' = \frac{T(H \setminus e; x, y)}{T_L(H; x, y)} \qquad y' = \frac{T(H/e; x, y)}{T_C(H; x, y)}.$$

Let $n = |V(H)|$, $m = |E(H)|$ and $k = k(E(H))$. Brylawski's tensor product formula states that

$$T(G \otimes_e H; x, y) = T_C(H; x, y)^{m-n+k} T_L(H; x, y)^{n-k} T(G; x', y').$$

[3] Note that this is different from the standard tensor product for graphs.

## 3 Reducing along the curve $H_\alpha$

In this section we describe how we can lift hardness results from a single point $(a, b) \in H_\alpha$ to the whole curve $H_\alpha$. We summarize the results from this section in the following theorem.

▶ **Theorem 3.1.** *Let $(a, b) \in \mathbb{C}^2$. Also let $T(G; x, y)$ be the Tutte polynomial of $G$ and $\alpha := (a-1)(b-1)$. There exists a polynomial time reduction from computing $T$ on $(a, b)$ for graphs of given tree-, path- or cutwidth, to computing $T$ along $H_\alpha$ for graphs width the following with parameters.*

- *If $|a| \notin \{0, 1\}$ or if $|b| \notin \{0, 1\}$ and $a \neq 0$, then the treewidth remains $\mathrm{tw}(G)$. The cutwidth and pathwidth become at most $\mathrm{ctw}(G) + 2$ and $\mathrm{pw}(G) + 2$ respectively.*
- *If $|b| \notin \{0, 1\}$ and $a = 0$, then the treewidth remains $\mathrm{tw}(G)$. The pathwidth becomes at most $\mathrm{pw}(G) + 2$ and the cutwidth becomes at most $2\,\mathrm{ctw}(G)$.*
- *If $|a|, |b| \in \{0, 1\}$, then the treewidth remains $\mathrm{tw}(G)$. The pathwidth becomes at most $\mathrm{pw}(G) + 2$ and the cutwidth becomes at most $12\,\mathrm{ctw}(G)$.*

Theorem 3.1 lets us lift both algorithms and lower bounds from a point $(a, b)$ to the whole curve $H_\alpha$. While our main theorem only requires Theorem 3.1 to be stated for integer valued points, we will state it as the most general version we can prove. We note that for Case 1. of Theorem 1.1, we do not care too much about constant multiplicative factors in the cutwidth, since we have an ETH bound of the form $\mathrm{ctw}(G)^{o(\mathrm{ctw}(G))}$. For Case 2. we only need the bounds on the treewidth and pathwidth. Thus the blowup in the cutwidth is only relevant for Case 3.. In this case the only integer valued points that fall under the third item of Theorem 3.1 are $(-1, 0), (0, -1)$ and $(-1, -1)$. These are all special points, which means that this item is not relevant for Case 3..

In our proofs we will make use of the following transformations.

▶ **Definition 3.2** ([15])**.** *Let $G$ be a simple graph. We define the $k$-*stretch *${}^kG$ of $G$ as the graph obtained by replacing every edge by a path of length $k$. We define the $k$-*thickening *${}_kG$ of $G$ as the graph obtained by replacing every edge by $k$ parallel edges.*

A new variant we introduce to keep the cutwidth low is defined as follows:

▶ **Definition 3.3.** *We define the* insulated $k$-thickening *${}_{(k)}G$ as the graph obtained by replacing every edge by a path of length 3 and then replacing the middle edge in each of these paths by $k$ parallel edges.*

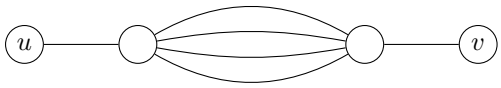



**Figure 3** The result of applying the insulated 4-thickening to an edge between $u$ and $v$.

### 3.1 Effect on width parameters

We give three lemmas that show how these transformations effect the parameters we use.

▶ **Lemma 3.4** (†)**.** *Let $G$ be a graph. Then we have that $\mathrm{tw}({}^kG) \leq \mathrm{tw}(G)$, $\mathrm{tw}({}_kG) \leq \mathrm{tw}(G)$ and $\mathrm{tw}({}_{(k)}G) \leq \mathrm{tw}(G)$.*

▶ **Lemma 3.5** (†)**.** *Let $G$ be a graph. Then we have that $\mathrm{pw}({}^kG) \leq \mathrm{pw}(G) + 2$, $\mathrm{pw}({}_kG) \leq \mathrm{pw}(G)$ and $\mathrm{pw}({}_{(k)}G) \leq \mathrm{pw}(G) + 2$.*

▶ **Lemma 3.6** (†). *Let $G$ be a graph. Then we have that* $\mathrm{ctw}({}^{k}G) \leq \mathrm{ctw}(G)$, $\mathrm{ctw}({}_{k}G) \leq k\,\mathrm{ctw}(G)$ *and* $\mathrm{ctw}({}_{(k)}G) \leq \mathrm{ctw}(G) + k - 1$.

We remark that the only significant blowup is that of the cutwidth, when applying the $k$-thickening. We will therefore limit our use of this transformation as much as possible.

## 3.2 Reductions

We now discuss the proof of Theorem 3.1. In the full version in the appendix we split the theorem into multiple separate cases. Here, we only give one case as a representative sample:

▶ **Lemma 3.7.** *Let $(a, b) \in \mathbb{C}^2$ be a point with $|a| \notin \{0, 1\}$. Also let $T(G; x, y)$ be the Tutte polynomial of $G$ and $\alpha := (a-1)(b-1)$. There exists a polynomial time reduction from computing $T$ on $(a, b)$ for graphs of given tree-, path- or cutwidth, to computing $T$ along $H_\alpha$ for graphs with the following with parameters. The treewidth and cutwidth remain* $\mathrm{tw}(G)$ *and* $\mathrm{ctw}(G)$ *respectively. The pathwidth becomes at most* $\mathrm{pw}(G) + 2$.

We prove this lemma using essentially the same proof as given in [15]. Note that in our setting we use Lemmas 3.4, 3.5 and 3.6 to ensure that relevant parameters are not increased by the operations we perform.

**Proof.** By Brylawski's tensor product formula [6], we find the following expression for the $k$-stretch of the graph G

$$(1 + a + \cdots + a^{k-1})^{k(E)} T\left(G; a^k, \frac{b + a + \cdots + a^{k-1}}{1 + a + \cdots + a^{k-1}}\right) = T({}^{k}G; a, b). \tag{3}$$

Note that

$$a^k - 1 = (1 + a + \cdots + a^{k-1})(a - 1)$$

and

$$\frac{b + a + \cdots + a^{k-1}}{1 + a + \cdots + a^{k-1}} - 1 = \frac{b - 1}{1 + a + \cdots + a^{k-1}}.$$

We find that the point on which we evaluate $T(G)$ in (3) also lies on $H_\alpha$.

By examining the formula for the Tutte polynomial, we find that for $n = |V(G)|$ the degree of the Tutte polynomial is at most $n^2 + n$. By choosing $k = 0, \ldots, n^2 + n$, since $|a| \notin \{0, 1\}$, we can find $T(G; x, y)$, for $n^2 + n + 1$ different values of $(x, y) \in H_\alpha$. By lemma 2.1, we can now interpolate the univariate restriction

$$T_\alpha(G; t) = T\left(G; \frac{\alpha}{t} + 1, t + 1\right).$$

of $T(G)$ along $H_\alpha$.

Note that by Lemmas 3.4 and 3.6 the $k$-stretch preserves both the cutwidth and the treewidth of the graph and by Lemma 3.5 the pathwidth increases by a constant additive factor. We find that any fine-grained parameterized lower bound for $H_\alpha$ extends to points $(a, b)$. ◀

# 4 Counting forests

In this section we consider the problem of counting the number of forests in a graph. This problem corresponds to the point $(2, 1)$ and thus by Theorem 3.1 any bounds found for this problem can be lifted to the whole curve $H_0^y$.

We trivially get the following lower bound from existing bounds on the non-parameterized version of the problem [12].

▶ **Theorem 4.1.** *Computing the Tutte polynomial along the curve $H_0^y$ cannot be done in time $2^{o(\mathrm{ctw}(G))}n^{O(1)}$, unless #ETH fails.*

To complement this lower bound, we give an algorithm to count the number of forests in a graph $G$ in $c^{\mathrm{tw}(G)}$ time. The algorithm uses a rank based approach, the runtime of which depends on the rank of the so called *forest compatibility matrix*. We start by introducing this matrix and examining its rank.

## 4.1 Notation

We will use the notation $[n] = \{1, \ldots, n\}$. Unless stated otherwise, we will assume the set $[n]$ to be ordered. We write $\pi \vdash S$ to indicate that $\pi$ is a partition of $S$.

We will consider matrices indexed by partitions. We will write $M[\pi, \rho]$ for the element in the row corresponding to $\pi$ and the column corresponding to $\rho$. We will write $M[\pi]$ for the vector containing all elements in the row corresponding to $\pi$.

## 4.2 Rank bound

In this section we prove the following theorem, for the so called *forest compatibility matrix* $F_n$.

▶ **Theorem 4.2.** *The rank of $F_n$ is at most $C_n$, the $n^{th}$ Catalan number. In particular* $\mathrm{rank}(F_n) = O(4^n n^{-3/2})$

Before we can define the forest compatibility matrix, we first need the following definitions.

▶ **Definition 4.3.** *We say that a boundaried graph $G = ([n] \cup V, E)$, with boundary $[n]$, is a* representative forest *for a partition $\pi \vdash [n]$, if for every $S \in \pi$ there is some connected component $C \subseteq V(G)$ such that $C \cap [n] = S$.*

*Given two boundaried graphs $G$ and $H$, both with boundary $B$, we define the* glue $G \oplus H$ *of $G$ and $H$ as follows. First take the disjoint union of $G$ and $H$. Then identify each $v \in B$ in $G$ with its analogue in $H$.*

This definition shows how one can relate forests and partitions. Throughout the section we will mostly consider partitions as they capture only the information we need. The following definition elaborates on this by lifting the concept of cycles in a clue of two trees to a cycle inducet by two partitions.

▶ **Definition 4.4.** *Let $\pi, \rho \vdash [n]$ and let $G_\pi$ and $G_\rho$ be representative forests of $\pi$ and $\rho$ respectively. We say that $\pi$ and $\rho$* induce a cycle *if $G_\pi \oplus G_\rho$ contains a cycle.*

It is not hard to see that it does not matter which representatives $G_\pi$ and $G_\rho$ we choose, since one only needs to know the connected components on $[n]$. This means that this definition is indeed well-defined. For this same reason, in the following definition, we only need a row and column for each partition of the separator.

▶ **Definition 4.5.** *We define the* forest compatibility matrix $F_S$ *of a set* $S$ *by*

$$F_S[\pi, \rho] := \begin{cases} 0 & \text{if } \pi \text{ and } \rho \text{ induce a cycle} \\ 1 & \text{otherwise} \end{cases}$$

*for any* $\pi, \rho \vdash S$*. We will write* $F_n := F_{[n]}$*.*

Finally we will need the following definition to bound the rank of the forest compatibility matrix.

▶ **Definition 4.6.** *We say that two sets* $A, B \in \pi$ *are* crossing *on an ordering* $<$*, if there are* $a_1, a_2 \in A$ *and* $b_1, b_2 \in B$ *such that* $a_1 < b_1 < a_2 < b_2$ *or* $b_1 < a_1 < b_2 < a_2$*. If a partition contains two crossing sets, we refer to it as a* crossing partition*.*

Throughout this section it will sometimes be convenient to think of the ordering as a permutation.

The general idea behind the proof of Theorem 4.2 is to show that any partition can be "uncrossed", i.e. its row in $F_n$ can be written as a linear combination of rows, corresponding to non-crossing partitions.

### 4.2.1 Manipulating partitions

For the proof of Theorem 4.2 we will need the following operations, which will allow us to manipulate partitions by contracting an expanding intervals and projecting down to subsets of the ground set.

▶ **Definition 4.7.** *An* interval *is a subset* $I \subseteq [n]$ *of consecutive numbers, i.e. there is no* $b \notin I$ *such that* $a_1 < b < a_2$ *for some* $a_1, a_2 \in I$*. Given an interval* $I$ *and a partition* $\pi$ *of* $[n]$*, we define the* contraction $\pi -_i I$ *of* $\pi$ *by* $I$ *as the partition of the set* $[n] -_i I := ([n] \cup \{i\}) \setminus I$ *given by we merging all sets that intersect* $I$ *and replacing* $I$ *by a single element* $i$*, i.e.*

$$\pi -_i I := \{S \in \pi : S \cap I = \emptyset\} \cup \left\{ \left( \bigcup \{S \in \pi : S \cap I \neq \emptyset\} \cup \{i\} \right) \setminus I \right\}.$$

*If we have an ordering on* $[n]$*, we place* $i$ *in the same place in the ordering as* $I$*, that is for any* $a \in [n] \setminus I$ *and* $b \in I$*, we have* $a < b$ *if and only if* $a < i$*.*

*We define the* blowup $\pi +_i I$ *of* $\pi$ *by* $I$ *as the partition of the set* $[n] +_i I := ([n] \cup I) \setminus \{i\}$*, given by adding all elements of* $I$ *to the set that contains* $i$ *and then removing* $i$*, i.e.*

$$\pi +_i I := \{S \in \pi : i \notin S\} \cup \{(S \setminus \{i\}) \cup I : i \in S\}.$$

*Again we place* $I$ *in the same place in the ordering as* $i$*.*

We will sometimes abuse notation and refer to $[n] -_i I$ as simply $[n']$ for $n' = n - |I| + 1$.

We now turn our attention to a number of useful lemmas. The first lemma intuitively says that if we contract an interval contained in some partition, then any decomposition of the resulting smaller partition gives the same decomposition of the larger partition.

▶ **Lemma 4.8.** *Let* $\pi$ *be a partition of* $[n]$ *and let* $I$ *be an interval such that* $I \subseteq S \in \pi$*. We set* $n' = n - |I| + 1$*. Suppose that for some set of partitions* $\mathcal{R}$ *of* $[n']$*, we have* $F_{n'}[\pi -_i I] = \sum_{\rho \in \mathcal{R}} a_\rho F_{n'}[\rho]$*. Then* $F_n[\pi] = \sum_{\rho \in \mathcal{R}} a_\rho F_n[\rho +_i I]$*.*

**Proof.** Let $\chi$ be some partition of $[n]$. Note that if $|S' \cap I| \geq 2$ for some $S' \in \chi$, we have that $F_n[\pi, \chi] = F_n[\rho +_i I, \chi] = 0$. Thus we may assume that $\chi$ contains no such sets. Also note that if there is some cycle that only requires $I$ and not the rest of $S$, then again we have that $F_n[\pi, \chi] = F_n[\rho +_i I, \chi] = 0$. Thus we may assume that any cycle induced by $\chi$ and $\pi$ that has a set that intersects $I$, also requires a set that intersects $S \setminus I$, but not $I$.

We now claim that for $\chi$ with the above assumptions we have $F_n[\rho +_i I, \chi] = F_n[\rho, \chi -_i I]$ for any $\rho$. This would immediately imply that for such $\chi$

$$F_n[\pi, \chi] = F_n[\pi -_i I, \chi -_i I] = \sum_{\rho \in R} a_\rho F_n[\rho, \chi -_i I] = \sum_{\rho \in R} a_\rho F_n[\rho +_i I, \chi],$$

which proves the lemma.

First note that if $\rho$ and $\chi -_i I$ induce a cycle, that does not involve $i$, then $\rho +_i I$ and $\chi$ also induce that same cycle and vice versa.

Now suppose that $\rho +_i I$ and $\chi$ induce a cycle involving $I$, then there is some $S'$ in the cycle that intersects $I$. By assumption there is also some set $S'' \in \chi$ in the cycle, that intersects $S \setminus I$, but not $I$. W.l.o.g. the cycle does not loop back on itself and thus these sets are the only two in the cycle that intersect $S$. Note that $S'$ gets merged into the set containing $i$, but $S''$ does not. Since the rest of the cycle does not involve $I$, it is unaffected and thus the cycle remains intact after contraction.

In the reverse direction we assume that $\rho$ and $\chi -_i I$ induce a cycle involving $i$, then it is clear to see that this cycle survives after blowing up $i$, using one of the sets in $\chi$ that intersect $I$. This proves the claim and thus the lemma. ◀

This next lemma intuitively says that if we project our partition to a subset of the ground set, then any decomposition of the resulting smaller partition gives the same decomposition of the larger partition.

▶ **Lemma 4.9.** *Let $\pi$ be a partition of $[n]$ and let $n' < n$. Suppose that for some set of partitions $\mathcal{R}$ of $[n']$, we have $F_{n'}[\pi|_{[n']}] = \sum_{\rho \in \mathcal{R}} a_\rho F_{n'}[\rho]$, then $F_n[\pi] = \sum_{\rho \in \mathcal{R}} a_\rho F_n[\rho \sqcup \pi|_{[n] \setminus [n']}]$.*

**Proof.** Let $\chi$ be some partition of $[n]$. If $\chi$ and $\pi|_{[n] \setminus [n']}$ induce a cycle, then the statement trivially holds. In the rest of the proof we will therefore assume that any cycle induced by $\chi$ and $\pi$ requires the use of $\pi|_{[n']}$.

We first define an equivalence relation $\sim$ on $[n]$ by defining two elements to be equivalent if they are either in the same set of $\chi$ or in the same set of $\pi|_{[n] \setminus [n']}$. We then complete this to a full equivalence relation. We now define the partition $\chi'$ of $[n']$ as the set of equivalence classes of $\sim$, restricted to $[n']$.

We claim that $F_n[\rho \sqcup \pi|_{[n] \setminus [n']}, \chi] = F_{n'}[\rho, \chi']$ for any $\rho$, which would immediately imply that

$$F_n[\pi, \chi] = F_{n'}[\pi|_{[n']}, \chi'] = \sum_{\rho \in \mathcal{R}} a_\rho F_{n'}[\rho, \chi'] = \sum_{\rho \in \mathcal{R}} a_\rho F_n[\rho \sqcup \pi|_{[n] \setminus [n']}, \chi]$$

which proves the lemma.

Suppose that $\rho \sqcup \pi|_{[n] \setminus [n']}$ and $\chi$ induce some cycle. Since the cycle must pass through $[n']$, there must be some path from one element of $[n']$ to another, induced by $\rho \sqcup \pi|_{[n] \setminus [n']}$ and $\chi$. Since all elements in this path are equivalent, this path must lie entirely inside of a set $S' \in \chi'$ and thus replacing such a path with $S'$ results in a cycle induced by $\rho$ and $\chi'$. Note that if a cycle only requires sets from $\pi|_{[n']}$, this operation results in a single set $S'$ from $\chi'$ in the new cycle. However, since any set involved in the old cycle must contain at least two elements in the path, that set together with $S'$ induces a cycle.

Similarly, in the reverse direction we take a cycle induced by $\rho$ and $\chi'$ and blow up any sets of $\chi'$ into a path in the corresponding connected component to find a cycle induced by $\rho \sqcup \pi|_{[n]\setminus[n']}$ and $\chi$. ◀

The following two lemmas help ensure that our operations do not introduce new crossings. The first of the two lemmas shows us that we can safely contract an interval, so long as it is contained in a set of the partition.

▶ **Lemma 4.10** (†). *Let $I \subseteq [n]$ be an interval of $[n]$. Let $\pi$ be a non-crossing partition of $[n] -_i I$. Then $\pi +_i I$ is also non-crossing.*

This next lemma shows us that, in our setting, projection is safe, as long as we do not forget any elements of sets that cross one another.

▶ **Lemma 4.11** (†). *Let $\pi \vdash [n]$ be a partition such that only $A, B \in \pi$ cross each other and all other pairs of sets in $\pi$ are non-crossing. Then for a non-crossing partition $\rho$ of $A \cup B$ we have that $\rho \cup \pi|_{[n]\setminus(A\cup B)}$ is non-crossing.*

### 4.2.2 Proof of the rank bound

With Lemmas 4.8, 4.9, 4.10 and 4.11 in hand, we are now ready to describe the main uncrossing operation.

▶ **Lemma 4.12.** *Let $\pi$ be a non-crossing partition on an ordering $p$. In time $O(n)$ we can find constants $c_\rho$, such that $F_n[\pi] = \sum_{\rho\in\mathcal{N}} c_\rho F_n[\rho]$, where $\mathcal{N}$ is the set of partitions that are non crossing on $p \circ (i, i+1)$.*

**Proof.** Throughout the proof, we will consider the partition $\pi$ on the ordering $p \circ (i, i+1)$. We first note that since $\pi$ is non-crossing on $p$, any crossing of $\pi$ must involve both $i$ and $i+1$. Let $i \in A \in \pi$ and $i+1 \in B \in \pi$. If $A = B$, then $\pi$ is non-crossing and thus we may assume that $A \neq B$. Note that $\pi|_{A\cup B}$, when viewed as a partition of $A \cup B$, consists of either 4 or 5 intervals which alternate between $A$ and $B$. Define $\pi'$ as the partition given by contracting these intervals. We find that $\pi'$ is a partition on $n'$ elements, where either $n' = 4$ or $n' = 5$ elements, with intervals of size 1 (see Figure 4).

We can explicitly construct the forest compatibility matrices for $n' \in \{4, 5\}$ and check that the non-crossing partitions give a basis. With this paper we provide a MATLAB script that verifies this. Thus we can write

$$F_{n'}[\pi'] = \sum_{\rho\in\mathcal{R}} c_\rho F_{n'}[\rho],$$

where $\mathcal{R}$ is the set of non-crossing partitions of $[n']$. By Lemma 4.8 we find that

$$F_{A\cup B}[\pi|_{A\cup B}] = \sum_{\rho\in\mathcal{R}} c_\rho F_{A\cup B}[\rho +_{i_1} I_1 + \cdots +_{i_{n'}} I_{n'}].$$

By Lemma 4.10 each $\rho +_{i_1} I_1 + \cdots +_{i_{n'}} I_{n'}$ is still non-crossing. By Lemma 4.9 we find

$$F_n[\pi] = \sum_{\rho\in\mathcal{R}} c_\rho F_{A\cup B}[(\rho +_{i_1} I_1 + \cdots +_{i_{n'}} I_{n'}) \cup \pi|_{[n]\setminus(A\cup B)}].$$

By Lemma 4.11 each $(\rho +_{i_1} I_1 + \cdots +_{i_{n'}} I_{n'}) \cup \pi|_{[n]\setminus(A\cup B)}$ is still non-crossing. We conclude that $F_n[\pi]$ can be written as a linear combination of rows corresponding to non-crossing partitions.

Note that we can construct $\pi'$ in $O(n)$ time. We then find the $c_\rho$ in $O(1)$ time and reconstruct the $(\rho +_{i_1} I_1 + \cdots +_{i_{n'}} I_{n'}) \cup \pi|_{[n]\setminus(A\cup B)}$ in $O(n)$ time. ◀

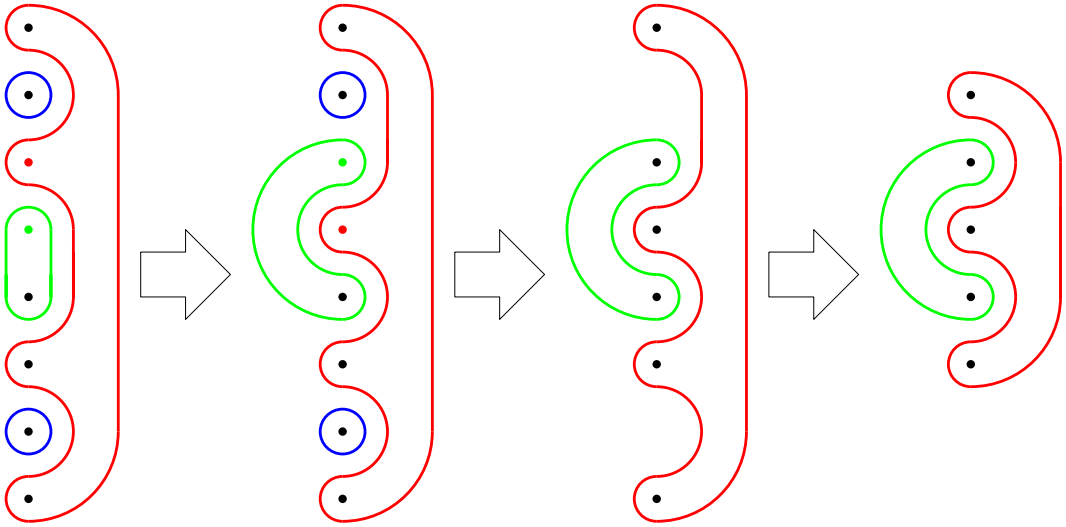


**Figure 4** From left to right, these are examples of $\pi$ before the swap, $\pi$ after the swap, $\pi|_{A\cup B}$ and $\pi'$.

By repeatedly applying Lemma 4.12, we can prove the following theorem.

▶ **Theorem 4.13.** *The rows corresponding to non-crossing partitions span a row basis of the forest compatibility matrix $F_n$.*

**Proof.** Let $\pi$ be a partition of $[n]$ such that we can turn it into a non-crossing partition by swapping two consecutive elements $i$ and $i+1$ in the order of $[n]$. By Lemma 4.12 we can write the row $F_n[\pi]$ corresponding to $\pi$ as a linear combination of rows corresponding to non-crossing partitions of $[n]$. This shows that, for $B_p$ the set of rows corresponding to non-crossing partitions on $p$, we have $B_{p\circ(i,i+1)} \subseteq \text{span}(B_p)$. Since every partition is non-crossing for some permutation and every permutation can be decomposed into 2-cycles on consecutive elements, this implies that every row can be written as a linear combination of rows corresponding to non-crossing partitions on some fixed ordering $p$. ◀

From this we immediately find a proof for Theorem 4.2.

**Proof of Theorem 4.2.** By Theorem 4.13 the non-crossing partitions form a basis of $F_n$. Since there are $C_n$ such partitions we find $\text{rank}(F_n) \leq C_n$. ◀

## 4.3 Algorithm

We will now describe the algorithm for counting forests. We first define the dynamic programming table and the notion of representation. For details on how to compute the table entries, see the full version [20].

▶ **Definition 4.14.** *Let $G$ be a graph and let $(\mathbb{T}, (B_x)_{x\in V(D)})$ be a tree/path decomposition of $G$. Recall that $G_x$ is defined as the graph induced by the union of all bags, whose nodes are descendants of $x$ in $\mathbb{T}$. We define the dynamic programming table $\tau$ by*

$$\tau_x[\pi] := |\{X \subseteq E(G_x) : (V, X) \text{ is acyclic},\\ \forall u, v \in B_x \text{ there is a path in } (V, X) \text{ from } u \text{ to } v \text{ iff } \exists S \in \pi \text{ s.t. } u, v \in S\}|$$

In other words, the table entry $\tau_x[\pi]$ counts the number of forests in $G_x$ whose connected components agree with $\pi$. In the rest of this section, we will refer to the number of nonzero entries $\tau_x[\pi]$ in a 'row' $\tau_x$ of the dynamic programming table as the support of $\tau_x$, written $\text{supp}(\tau_x)$. Our aim will be to ensure that the support of our rows remains contained in the entries corresponding to non-crossing partitions for some ordering on the bag $B_x$. This is captured in the following definition.

▶ **Definition 4.15.** *We say a vector $a$, indexed by partitions, is* reduced on an ordering $p$, *if $a_\pi = 0$ for any partition $\pi$ that is crossing for $p$.*

In order to ensure that we do not lose any relevant information we will reduce our rows, while retaining the following property for the matrix $F_{B_x}$.

▶ **Definition 4.16.** *Given a matrix $M$, we say that a vector $a$ $M$-*represents *a vector $b$ if $Ma = Mb$*

In the full version of the paper[20], we describe a dynamic programming algorithm that works with reduced rows, rather than the whole table. It does so by alternating between reducing the current row in the table and computing the next row. This allows us to work with a table where the rows effectively have size $\text{rank}(F_{\text{tw}})$ (or $\text{rank}(F_{\text{pw}})$). Doing so, we establish the following:

▶ **Theorem 4.17** (†). *There exists an algorithm that, given a graph $G$ with a path decomposition of width $\text{pw}(G)$, computes the number of forests in the graph in time $4^{\text{pw}(G)}n^{O(1)}$.*

▶ **Theorem 4.18** (†). *There exists an algorithm that, given a graph $G$ with a tree decomposition of width $\text{tw}(G)$, computes the number of forests in the graph in time $64^{\text{tw}(G)}n^{O(1)}$.*

# 5 Other cases

In this section we handle the remaining cases mentioned in Theorem 1.1.

## 5.1 The curve $H_2$

The curve $H_2$ is equivalent to the partition function of the Ising model. Both our proofs for the upper and lower bound on the complexity will make use of this fact.

▶ **Theorem 5.1.** *Computing the Tutte polynomial along the curve $H_2$ cannot be done in time $(2-\epsilon)^{\text{ctw}(G)}n^{O(1)}$, unless SETH fails.*

**Proof Sketch.** Using known equivalences we first reduce #MaximumClosedSubgraphs to the problem of computing the Tutte polynomial along the curve $H_2$. We then apply a simple transformation, based on a similar argument by [17] to ensure the graph only has odd degree vertices. It then suffices to note that on graphs with only odd degree vertices, the complement of a perfect matching is a maximum closed subgraph and thus #MaximumClosedSubgraphs is equivalent to #PerfectMatchings on such graphs. This allows us to lift an existing lower bound from [8] on #PerfectMatchings to #MaximumClosedSubgraphs. ◀

We also show in the full version that this lower bound can be matched with a tight upper bound. The proof uses dynamic programming combined with subset convolution [3, 9].

▶ **Theorem 5.2.** *Let $G$ be a graph with a given tree decomposition of width $\text{tw}(G)$. There exists an algorithm that computes $T(G;a,b)$, for $(a,b) \in H_2$, in time $2^{\text{tw}(G)}n^{O(1)}$.*

## 5.2 The curve $H_0^x$

The curve $H_0^x$ contains the point $(1, 2)$, which counts the number of connected edgesets of a connected graph. Using existing results this gives an ETH lower bound which matches the running time of the algorithm mentioned in theorem 1.1.

▶ **Theorem 5.3.** *Let $0 < \alpha < 1$. Computing the Tutte polynomial along the curve $H_0^x$ cannot be done in time $(\alpha \operatorname{ctw}(G) - \epsilon)^{(1-\alpha)\operatorname{ctw}(G)/2} n^{O(1)}$, unless SETH fails.*

**Proof.** In [13] a lower bound of $p^{\operatorname{ctw}(G)}$ is found for counting connected edgesets modulo $p$. In the reduction the authors reduce to counting essentially distinct $q$-coloring modulo $p$, with cutwidth $\operatorname{ctw}(G) + q^2$ and $p = q$. Thus we find a lower bound of $p^{\operatorname{ctw}(G)-p^2} = (\alpha \operatorname{ctw}(G))^{(1-\alpha)\operatorname{ctw}(G)/2}$ for $p = (\alpha \operatorname{ctw}(G))^{1/2}$. ◀

## 5.3 The curve $H_q$ for $q \in \mathbb{Z}_{\geq 3}$

These curves contain the points $(1 - q, 0)$, which count the number of $q$-colorings. Using previous results and a folklore algorithm, we find matching upper and lower bounds for these points and thus for the whole curves.

▶ **Theorem 5.4.** *Let $q \in \mathbb{Z}_{\geq 3}$. Computing the Tutte polynomial along the curve $H_q$ cannot be done in time $(q - \epsilon)^{\operatorname{ctw}(G)} n^{O(1)}$, unless SETH fails.*

**Proof.** Note that $H_q$ contains the point $(1 - q, 0)$. Computing the Tutte polynomial on this point is equivalent to counting the number of $q$-colorings of the graph $G$.

By choosing a modulus $p > q$ we can apply the results from [13] to find a lower bound of $q^{\operatorname{ctw}(G)}$ on the time complexity of counting $q$-colorings modulo $p$. This lower bound clearly extends to general counting. ◀

▶ **Theorem 5.5.** *Let $G$ be a graph with a given tree decomposition of width $\operatorname{tw}(G)$ and $q \in \mathbb{Z}_{\geq 3}$. There exists an algorithm that computes $T(G; a, b)$ for $(a, b) \in H_q$ in time $q^{\operatorname{tw}(G)} n^{O(1)}$.*

This theorem is a direct consequence of combining Theorem 3.1 with the following folklore result:

▶ **Theorem 5.6** (Folklore). *Let $G$ be a graph with a given tree decomposition of width $\operatorname{tw}(G)$ and $q \in \mathbb{Z}_{\geq 3}$. There exists an algorithm that computes the number of $q$-colorings of $G$ in time $q^{\operatorname{tw}(G)} n^{O(1)}$.*

## 5.4 The curve $H_{-q}$ for $q \in \mathbb{Z}_{>0}$

These curves contain the points $(1 + q, 0)$. Using the same results we used to prove theorem 5.4 and exploiting the fact these results hold for modular counting, we find an ETH lower bound which matches the running time of the algorithm mentioned in theorem 1.1.

▶ **Theorem 5.7.** *Let $q \in \mathbb{Z}_{>0}$. Computing the Tutte polynomial along the curve $H_{-q}$ cannot be done in $\operatorname{ctw}(G)^{o(\operatorname{ctw}(G))}$ time, unless ETH fails.*

**Proof.** Like mentioned earlier $H_{-q}$ contains the point $(1 + q, 0)$. For a prime $p > q$ we have that $T(G; 1 + q, 0) \equiv_p T(G; 1 + q - p, 0)$. This means that computing the Tutte polynomial modulo $p$ at the point $(1 + q, 0)$ is equivalent to counting the number of $p - q$-colorings of $G$ modulo $p$. Since $q > 0$ and $p > q$ we find that $0 < p - q < p$ and thus as before, by [13], we find a lower bound of $(p - q)^{\operatorname{ctw}(G)}$. Since the cutwidth of the construction in [13] is $O(n + rp^{r+2})$ for some $r$ dependant on $p - q$ and $\epsilon$. We find that there is no algorithm running in time $O((p - q - \epsilon)^{\operatorname{ctw}(G) - rp^{r+2}}) = O((\alpha \operatorname{ctw}(G) - \epsilon)^{\operatorname{ctw}(G)(1-\alpha)/(r+2)})$, where $p - q = (\alpha \operatorname{ctw}(G))^{1/(r+2)}$. ◀

## 6 Conclusion

In this paper we gave a classification of the complexity, parameterized by treewidth/pathwidth/cutwidth, of evaluating the Tutte polynomial at integer points into either computable

- in polynomial time,
- in $\mathrm{tw}^{O(\mathrm{tw})}\, n^{O(1)}$ time but not in $\mathrm{ctw}^{o(\mathrm{ctw})}\, n^{O(1)}$ time,
- in $q^{\mathrm{tw}} n^{O(1)}$ time but not in $2^{o(\mathrm{ctw})}$ (and for many points not even in $r^{\mathrm{ctw}} n^{O(1)}$ time for some constants $q > r$),

assuming the (Strong) Exponential Time Hypothesis.

This classification turned out to be somewhat surprising, especially considering the asymmetry between $H_0^x = \{(x, y) : x = 1\}$ and $H_0^y = \{(x, y) : y = 1\}$ that does not show up in other classifications such as the ones from [5, 12, 15].

Our paper leaves ample opportunities for further research. First, we believe that our rank upper bound should have more applications for counting forests with different properties. For example, it seems plausible that it can be used to count all Feedback Vertex Sets in time $2^{O(\mathrm{tw})} n^{O(1)}$ or the number of spanning trees with $k$ components in time $2^{O(\mathrm{tw})} n^{O(1)}$. The latter result would improve over a result by Peng and Fei Wan [22] that show how to count the number of spanning forests with $k$ components (or equivalently, $n - k - 1$ edges) in $\mathrm{tw}^{O(\mathrm{tw})}\, n^{O(1)}$ time. We decided to not initiate this study in this paper to retain the focus on the Tutte polynomial.

Second, it would be interesting to see if our classification of the complexity of all points on $\mathbb{Z}^2$ can be extended to a classification of the complexity of all points on $\mathbb{R}^2$ (or even $\mathbb{C}^2$). Typically, evaluation at non-integer points can be reduced to integers points (leading to hardness for non-integer points), but we were not able to establish such a reduction without considerably increasing the width parameters.

Third, it would be interesting to see if a similar classification can be made when parameterized by the vertex cover number instead of treewidth/pathwidth/cutwidth. We already know that the runtime of $2^n n^{O(1)}$ by Björklund et al. [2] for evaluating the Tutte polynomial cannot be strengthened to a general $2^{O(k)} n^{O(1)}$ time algorithm where $k$ is the minimum vertex cover size of the input graph due to a result by Jaffke and Jansen [16], but this still leaves the complexity of evaluating at many other points open.

## References


1. Artur Andrzejak. An algorithm for the Tutte polynomials of graphs of bounded treewidth. *Discret. Math.*, 190(1-3):39–54, 1998. doi:10.1016/S0012-365X(98)00113-7.
2. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Computing the Tutte polynomial in vertex-exponential time. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 677–686. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.40.
3. Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets Möbius: fast subset convolution, 2006. doi:10.48550/arXiv.CS/0611101.
4. Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015. doi:10.1016/j.ic.2014.12.008.
5. Cornelius Brand, Holger Dell, and Marc Roth. Fine-grained dichotomies for the Tutte plane and Boolean #csp. *Algorithmica*, 81(2):541–556, 2019. doi:10.1007/s00453-018-0472-z.
6. Thomas H Brylawski. The Tutte polynomial. In *Matroid Theory and Its Applications: Lectures Given at the Centro Internazionale Matematico Estivo*. Springer Berlin, Heidelberg, Varenna (como), Italy, 1980. doi:10.1007/978-3-642-11110-5.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

8 Radu Curticapean and Dániel Marx. *Tight conditional lower bounds for counting perfect matchings on graphs of bounded treewidth, cliquewidth, and genus*, pages 1650–1669. Society for Industrial and Applied Mathematics, 2015. doi:10.1137/1.9781611974331.ch113.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. *J. ACM*, 65(3):12:1–12:46, 2018. doi:10.1145/3148227.

11 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

12 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlén. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Transactions on Algorithms*, 10(4):1–32, August 2014. doi:10.1145/2635812.

13 Carla Groenland, Isja Mannens, Jesper Nederlof, and Krisztina Szilágyi. Tight Bounds for Counting Colorings and Connected Edge Sets Parameterized by Cutwidth. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, pages 36:1–36:20, 2022. doi:10.48550/arXiv.2110.02730.

14 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

15 F. Jaeger, D. L. Vertigan, and D. J. A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 108(1):35–53, 1990. doi:10.1017/S0305004100068936.

16 Lars Jaffke and Bart M. P. Jansen. Fine-grained parameterized complexity analysis of graph coloring problems. *Discret. Appl. Math.*, 327:33–46, 2023. doi:10.1016/j.dam.2022.11.011.

17 Mark Jerrum. Two-dimensional monomer-dimer systems are computationally intractable. *Journal of Statistical Physics*, 48(1-2):121–134, July 1987. doi:10.1007/BF01010403.

18 John E. Krizan, Peter F. Barth, and M.L. Glasser. Phase transitions for the Ising model on the closed Cayley tree. *Physica A: Statistical Mechanics and its Applications*, 119(1):230–242, 1983. doi:10.1016/0378-4371(83)90157-7.

19 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. *SIAM J. Comput.*, 47(3):675–702, 2018. doi:10.1137/16M1104834.

20 Isja Mannens and Jesper Nederlof. A fine-grained classification of the complexity of evaluating the Tutte polynomial on integer points parameterized by treewidth and cutwidth. *CoRR*, To-appear, 2023. URL: https://arxiv.org/abs/2307.01046.

21 Steven D. Noble. Evaluating the Tutte polynomial for graphs of bounded tree-width. *Comb. Probab. Comput.*, 7(3):307–321, 1998. URL: http://journals.cambridge.org/action/displayAbstract?aid=46641.

22 Xin-Zhuang Chen Peng-Fei Wan. Computing the number of $k$-component spanning forests of a graph with bounded treewidth. *Journal of the Operations Research Society of China*, 7(2):385, 2019. doi:10.1007/s40305-019-00241-4.

# Matching Statistics Speed up BWT Construction

**Francesco Masillo**
Department of Computer Science, University of Verona, Italy

## Abstract

Due to the exponential growth of genomic data, constructing dedicated data structures has become the principal bottleneck in common bioinformatics applications. In particular, the Burrows-Wheeler Transform (BWT) is the basis of some of the most popular self-indexes for genomic data, due to its known favourable behaviour on repetitive data.

Some tools that exploit the intrinsic repetitiveness of biological data have risen in popularity, due to their speed and low space consumption. We introduce a new algorithm for computing the BWT, which takes advantage of the redundancy of the data through a compressed version of matching statistics, the *CMS* of [Lipták et al., WABI 2022]. We show that it suffices to sort a small subset of suffixes, lowering both computation time and space. Our result is due to a new insight which links the so-called insert-heads of [Lipták et al., WABI 2022] to the well-known run boundaries of the BWT.

We give two implementations of our algorithm, called `CMS-BWT`, both competitive in our experimental validation on highly repetitive real-life datasets. In most cases, they outperform other tools w.r.t. running time, trading off a higher memory footprint, which, however, is still considerably smaller than the total size of the input data.

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

**Keywords and phrases** Burrows-Wheeler Transform, matching statistics, string collections, compressed representation, data structures, efficient algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.83

**Supplementary Material** *Software (Source Code)*: `https://github.com/fmasillo/CMS-BWT`

**Acknowledgements** I want to thank Sara Giuliani for listening and discussing the preliminary ideas contained in this paper. I also want to thank Zsuzsanna Lipták for giving helpful feedback during the writing of this paper.

## 1 Introduction

The Burrows-Wheeler Transform (BWT) [6] is a reversible permutation of the characters of the input text that can be computed in linear time and space. It is closely related to the suffix array, a permutation of the indices of $T$ which is based on the lexicographic order of the suffixes. It is known that the BWT, when applied to repetitive texts, results in an easier-to-compress string, especially when using simple run-length encoding.

Another virtue of the BWT is that it can be used as a self-index to replace the original text. It can support pattern matching queries, more specifically, it counts how many times a pattern $P$ occurs in $T$ using time proportional to $|P|$. It can be incorporated into more elaborate indexes [11, 13, 14, 27] to support locating queries (finding the positions in $T$ where $P$ occurs) and more complex queries such as finding Maximal Exact Matches (MEMs) and Maximal Unique Matches (MUMs). MEMs and MUMs are of key importance, especially in the field of bioinformatics, where they are used for read alignment (e.g. MUMmer [23]). Widely used tools such as BowTie2 [18] and BWA [20] are based on the BWT for aligning short reads to a reference genome.

Nowadays, the amount of biological data that is publicly available is too big for most BWT construction algorithms. A key observation is that, even though the size of these datasets is massive, the information contained within them is highly redundant. This is why


tools such as `big-BWT` [5], `r-pfBWT` [26] and `grlBWT` [10] have emerged. These tools exploit the intrinsic repetitiveness of the input data to build large BWTs fast and in compressed space.

Recently, Lipták et al. [22] devised a variant of matching statistics [8] called *compressed matching statistics* (*CMS*). This data structure has been proven to be effective in expressing the redundancy of sets of highly similar strings and being used in the process of suffix sorting. Cunial et al. [9] work with a *compact* representation of matching statistics, i.e. a bitvector of length $2|P|$ storing the differentially encoded lengths of the matches. They also present several ways of compressing the bitvector based on both lossless and lossy methods. This representation is fundamentally different from that of [22] due to the fact that the *CMS*, and more prominently its enhanced version, packs more information than just the length of the matches. This additional information can then be used for other applications such as suffix sorting.

In this paper, we are going to show that the *CMS* of [22] not only can be used to build the generalized suffix array of a collection of strings but can also be highly useful in building large BWTs, due to the fact that it uses considerably less space than the input data. Experimental results show that our implementation, `CMS-BWT`, is competitive, if not better, than the state-of-the-art tools for constructing the BWT, although heavier on the space consumption side.

Our algorithm is based on a new insight that allows us to find the run boundaries of the BWT within special buckets of suffixes, which are closely connected to the fundamental element of the compressed matching statistics, the so-called *insert-heads*.

The paper is organized as follows. In Section 2, we give definitions and notations used in the remainder of the paper. Section 3 contains an overview of the compressed matching statistics. In Section 4, we present our contribution, describing the algorithm used for constructing the BWT. In Section 5, we describe details of our implementation and then report experimental results in Section 6. Finally, in Section 7, conclusions and future work are discussed.

## 2 Basics

Let $\Sigma$ be an ordered alphabet of size $\sigma$. A string $T$ over $\Sigma$ is a finite sequence of characters from $\Sigma$. The $i$th character of $T$ is denoted $T[i]$, its length is $|T| = n$, and $T[i..j]$ denotes the substring $T[i] \cdots T[j]$. If $i > j$, then $T[i..j]$ is the empty string $\varepsilon$. The suffix $T[i..] = T[i..n]$ is referred to as the $i^{th}$ suffix $suf_i(T)$, and $T[..i] = T[1..i]$ is the $i^{th}$ prefix $pref_i(T)$. When T is clear from the context, we write $suf_i$ for $suf_i(T)$.

We assume that the last character of $T$ is the sentinel character \$. It is set to be smaller than any other character in $\Sigma$ and appears only once as the end-of-string character.

The *suffix array* $SA$ of a string $T$ is a permutation of the set $\{1, \ldots, n\}$ such that $SA[i] = j$ if $suf_j(T)$ is the $i$th in lexicographic order among all suffixes. Numerous suffix array construction algorithms (SACAs) exist in the literature [25, 24, 1, 21, 15]. SA-IS [25] is by far the most popular linear time SACA, being both simple and fast in practice.

The *inverse suffix array* $ISA$ is the inverse permutation of $SA$, namely $ISA[SA[i]] = i$.

The *longest common prefix* (*lcp*) of a pair of strings $T$ and $S$ is the longest string $U$ which is the prefix of both $T$ and $S$. The *longest-common-prefix array* $LCP$ is another array closely related to the $SA$. It is given by: $LCP[1] = 0$, and for $i > 1$, $LCP[i]$ is the length of the longest common prefix of the two suffixes $suf_{SA[i-1]}$ and $suf_{SA[i]}$. This array can be computed in linear time, too [17].

The Burrows-Wheeler Transform BWT [6] is a reversible permutation of the input text $T$. It is defined as $\text{BWT}[i] = \$$ if $SA[i] = 1$, and $\text{BWT}[i] = T[SA[i] - 1]$ otherwise.

Let $R$ and $S$ be two strings. The *matching statistics MS* of $S$ w.r.t. $R$ is an array of length $|S|$ in which every entry is a pair of integers defined as follows. Fix $i$, let $U_i$ be the longest prefix of suffix $suf_i(S)$ which occurs as a substring in $R$. Then, entry $MS[i] = (p_i, \ell_i)$, where $p_i$ is an occurrence of $U_i$ in $R$, or $-1$ if $U_i = \varepsilon$, and $\ell_i = |U_i|$. We will call $U_i$ *matching factor* and the character $c_i = S[i + \ell_i]$ will be referred to as *mismatch character* of position $i$. We set the end-of-string character of $R$ to be smaller than the one of $S$ ($\# < \$$).

For an integer array $A$ of length $n$ and an index $i$, the previous and next smaller values are defined as follows: $PSV(A, i) = \max\{i' < i \ : \ A[i'] < A[i]\}$, $NSV(A, i) = \min\{i' > i \ : \ A[i'] < A[i]\}$. The minimum of the empty set is $-\infty$ and the maximum is $+\infty$. There exists a data structure of size $n \log(3 + 2\sqrt{2}) + o(n)$ bits that can be built in $\mathcal{O}(n)$ time and answers both *PSV* and *NSV* queries in constant time [12].

Given a set of integers $X$ and an integer $x$, the predecessor of $x$ is the largest element in $X$ less than or equal to $x$. In other words, $pred_X(x) = \max\{y \in X \ : \ y \leq x\}$. Predecessor queries can be answered in $\mathcal{O}(\log \log |X|)$ time using the y-fast trie data structure of Willard [29] which uses $\mathcal{O}(|X|)$ space.

Let $\mathcal{C} = \{S_1, S_2, \ldots, S_m\}$ be a collection of strings not necessarily distinct, i.e. $\mathcal{C}$ is a multiset. The total length of $\mathcal{C}$ will be denoted by $N$, where we use end-of-string characters to delimit the strings, i.e. $N = \sum_{d=1}^{m} |S_d| + m$. From now on, we will treat $\mathcal{C}$ as this concatenated string, slightly abusing notation.

Our problem is defined as follows:

**Problem Statement:** Given a string collection $\mathcal{C} = \{S_1, \ldots, S_m\}$ and a reference string $R$, compute the Burrows-Wheeler Transform BWT of $\mathcal{C}$.

The end-of-string character # of $R$ is assumed to be smaller than any of $\mathcal{C}$. Moreover, in our setting, we assume that each $S_i \in \mathcal{C}$ is highly similar to $R$.

## 3 Compressed Matching Statistics

Recently, the authors of [22] introduced a new data structure called Compressed Matching Statistics (*CMS*). This data structure exploits the redundancy of plain *MS*, where we have the following property: if $\ell_i > 0$, then $\ell_{i+1} \geq \ell_i - 1$. We can identify sequences of the form $(x, x - 1, x - 2, \ldots)$ where $x = \ell_i$, called *decrement runs*. A decrement run ends when $\ell_j > \ell_{j-1} - 1$, and $j$ is the starting position of a head. For an example see Figure 1.

▶ **Definition 1** (Compressed matching statistics, [22]). *Let $R, S$ be two strings over $\Sigma$, and MS be the matching statistics of $S$ w.r.t. $R$. The* compressed matching statistics (*CMS*) of $S$ w.r.t. $R$ *is a data structure storing $(j, MS[j])$ for each head $j$, and a predecessor data structure on the set of heads $H$.*

It was shown in [22] that it is possible to recover each individual value $MS[i]$ for any $i$ using the following formula: $MS[i] = (p_i + k, \ell_i - k)$, where $j = pred_H(i)$ and $k = j - i$. This can be done in $\mathcal{O}(\log \log \chi)$ time and $\mathcal{O}(\chi)$ space, where $\chi = |H|$.

It was shown in [22] that storing the matching statistics information only for heads leads to a compression ratio of up to 100 times on real-life data.

## 3.1 Enhanced Compressed Matching Statistics

In [22], the *CMS* was refined with additional information to get the *enhanced compressed matching statistics* (*eCMS*). Assuming that all characters occurring in $S$ also occur in $R$ at least once, the information of $p_i$ can be made more specific, namely, one can compute the position that a suffix from $S$ would have if it was present in $SA_R$ the $SA$ of $R$. This position is called *insert point* of $i$:

$$ip(i) = \begin{cases} 1 & \text{if } U_i = \varepsilon, \\ \max\{j \mid U_i \text{ is a prefix of } R[SA_R[j]..] \text{ and } R[SA_R[j]..] < U_i c\} & \text{if this set is non-empty,} \\ \min\{j \mid U_i \text{ is a prefix of } R[SA_R[j]..]\} & \text{otherwise.} \end{cases}$$

The first case is satisfied only for the end-of-string characters of the collection $\mathcal{C}$, because the sentinel character of $R$ is smaller than any other character (# < \$). In the other two cases, the insert point is the lexicographic rank of suffix $i$ among all suffixes of $R$. Suffix $i$ ideally points to the next smaller occurrence of $U_i$ in $R$, if it exists (case 2). Otherwise, it coincides with the smallest occurrence of $U_i$ in $R$ (case 3).

For the *eCMS*, the positions for which the *MS* information is saved are called *insert-heads* and are defined as follows: $j$ is an insert-head if $SA_R[ip(j)] \neq SA_R[ip(j-1)] + 1$. Some additional information is also stored in each insert-head: $c_i$, the mismatching character, and $x_i$, a boolean value associated with $c_i$. This value is set to be smaller (S = 0) if $c_i < R[SA[ip(i)] + \ell_i]$ or larger (L = 1) otherwise. Referring to the definition of $ip$, $x_i = 1$ whenever we are in case 2, $x_i = 0$ when we are in case 3.

▶ **Definition 2** (Enhanced compressed matching statistics, [22]). *Let $R, S$ be two strings over $\Sigma$. Define the* enhanced matching statistics *of $S$ w.r.t. $R$ as follows: for $1 \leq i \leq |S|$, let $ems(i) = (q_i, \ell_i, x_i, c_i)$, where $q_i = SA_R[ip(i)]$, $\ell_i$ is the length of the matching factor $U$ of $i$, $c_i$ is the mismatch character, and $x_i \in \{S, L\}$ indicates whether $U_i c_i$ is smaller (S) or greater (L) than $R[q_i..]$. The* enhanced compressed matching statistics (*eCMS*) of $S$ w.r.t. $R$ *is a data structure storing $(j, ems(j))$ for each insert-head $j$, and a predecessor data structure on the set of insert-heads $K$.*

The size of $K$ is denoted by $|K| = \kappa$. The time for recovering $MS[i]$ becomes $\mathcal{O}(\log\log\kappa)$, while the space becomes $\mathcal{O}(\kappa)$ [22].

By definition, the number of insert-heads is larger than the number of heads. Although in [22] the difference in numbers is noticeable, the compression effect is still very strong. For actual numbers see Section 6.2, more specifically Table 1.

For an example of *eCMS* refer to Figure 1.

## 3.2 Comparing two suffixes using eCMS

The additional information of insert-heads helps bucketing suffixes with respect to the insert point. We will call these buckets *insert-buckets*. Assessing the order of any two suffixes having different insert point has been proven in the following lemma:

▶ **Lemma 3** ([22]). *Let $1 \leq i, j \leq N$. If $ip(i) < ip(j)$, then $suf_i < suf_j$.*

On the other hand, when two suffixes belonging to the same insert-bucket are compared the following lemma refines the order:

▶ **Lemma 4** ([22]). *Let $1 \leq i, j \leq N$, and $ip(i) = ip(j)$.*

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R$ | C | A | T | T | A | G | A | T | T | A | G | # |
| $S$ | T | A | G | A | G | A | T | T | A | T | T | $ |
| $p_i$ | 4 | 5 | 6 | 5 | 6 | 7 | 8 | 9 | 2 | 3 | 4 | -1 |
| $\ell_i$ | 4 | 3 | 2 | 6 | 5 | 4 | 3 | 2 | 3 | 2 | 1 | 0 |
| head | ✓ | | | ✓ | | | | | ✓ | | | |
| $q_i$ | 4 | 5 | 6 | 5 | 6 | 2 | 3 | 4 | 7 | 8 | 9 | 12 |
| insert-head | ✓ | | | ✓ | | ✓ | | | ✓ | | | ✓ |
| $c_i$ | G | | | T | | T | | | $ | | | $ |
| $x_i$ | S | | | L | | L | | | S | | | L |

**Figure 1** An example for matching statistics and corresponding *CMS* and *eCMS*. In rows 1 and 2, we report $MS[i] = (p_i, \ell_i)$ of $S$ w.r.t. $R$. In row 3, we mark the starting positions of the heads (*CMS*). In row 4, for each index, we give the special position $q_i = SA_R[ip(i)]$, where $ip(i)$ is the insert point of $suf_i$ in $SA_R$. In row 5, we mark insert-heads (*eCMS*). In the last two rows, we complete the information stored in insert-heads, namely the mismatch character $c_i$ and $x_i$, the associated boolean value (S = smaller, L = larger).

1. *If* $\ell_i < \ell_j$ *and* $x_i = S$*, then* $suf_i < suf_j$.
2. *If* $\ell_i < \ell_j$ *and* $x_i = L$*, then* $suf_j < suf_i$.
3. *If* $\ell_i = \ell_j$ *and* $x_i = S$ *and* $x_j = L$*, then* $suf_i < suf_j$.
4. *If* $\ell_i = \ell_j$ *and* $x_i = x_j$ *and* $c_i < c_j$*, then* $suf_i < suf_j$.

To achieve the final correct order of two suffixes having the same insert-head information, in [22] it was suggested sorting only the insert-heads. Then, using the new rank for each head, the total order of two suffixes can be established.

If two arbitrary suffixes from $\mathcal{C}$ are being compared, one needs to perform two predecessor queries to get the insert-head of each suffix. This implies that the time spent for a single comparison is $\mathcal{O}(\log\log\kappa)$. As we will see in Section 4, we can avoid the predecessor queries when scanning the collection left to right, resulting in constant time comparisons. This is because we will only perform comparisons of suffixes of one insert-head at a time with other insert-heads of the same insert-bucket.

## 3.3 Computing the eCMS

We will use the procedure outlined in [22].

The data structures needed to compute the *eCMS* of $\mathcal{C}$ w.r.t. $R$ are the suffix array $SA_R$, the inverse suffix array $ISA_R$, the *LCP*-array $LCP_R$, and the *RMQ* data structure for *PSV-NSV* queries on $LCP_R$. Every data structure can be constructed in $\mathcal{O}(|R|)$ time and space.

This procedure takes $\mathcal{O}(N \log |R|)$ time and $\mathcal{O}(|R|)$ space and outputs the set of insert-heads of size $\mathcal{O}(\kappa)$.

To speed up the practical running time, we will use also the following proposed heuristic of [22]. Because we work with highly similar strings, it is common to have a singleton interval (an interval of size one) after the failure of a sequence of right extensions. A key insight is that also after the subsequent left contraction, the interval remains of size one. This means that the matching factor $U_i$ lies within a leaf branch in a hypothetical suffix tree of $R$. In order to detect these cases, we can compare $\ell_i$ to the maximum value in $LCP_R$. If $\ell_i - 1 > \max(LCP_R)$, then it means that there is no other suffix in $R$ with a prefix equal

to $U_i$. This means that we are in a leaf branch. Computing the left contraction is now equal to accessing $ISA[p_i + 1]$. This bypasses the $PSV$ and $NSV$ queries on $LCP_R$, avoiding the corresponding cache misses. Computing a single maximum can be too restricting for some datasets, so a refinement of this strategy is to divide $LCP_R$ into blocks and compute a maximum value for each of them.

The practical speedup can be of an order of magnitude when using this last strategy on sets of highly repetitive strings, as it was shown in [22].

## 4 Computing the BWT with enhanced Compressed Matching Statistics

In this section, we are going to outline the procedure used to compute the BWT of $\mathcal{C}$ using only data structures built on $R$ and the *eCMS* of $\mathcal{C}$ w.r.t. $R$.

We will use the following heuristic in order to speed up the computation of BWT($\mathcal{C}$): suffixes in text order between two insert-heads are preceded by the same character present in the reference. This can be intuitively explained by looking at the way *eCMS* is built: any position between two consecutive insert-heads is consecutive in text order both in $R$ and $\mathcal{C}$. Therefore, by knowing the insert point of each suffix we know what its position is in the previously computed $SA_R$, and consequently which is the preceding character stored in $\text{BWT}_R$. This insight tells us that we just have to "expand" $\text{BWT}_R$ based on the number of suffixes with the same insert-point while taking care of insert-heads. The suffixes corresponding to the starting positions of insert-heads are the only ones that need to be sorted inside each insert-bucket. See Figure 2 for an example.

▶ **Lemma 5.** *Let $suf_i$ and $suf_j$ be two suffixes of $\mathcal{C}$. If $ip(i) = ip(j)$ and the two suffixes are not the start of an insert-head, then $suf_i$ and $suf_j$ are preceded by the same character $c = R[SA[ip(i)] - 1]$, i.e. $\mathcal{C}[i-1] = \mathcal{C}[j-1] = c = R[SA[ip(i)] - 1]$.*

**Proof.** By assumption we know that $ip(i) = ip(j)$, therefore $SA[ip(i)] = SA[ip(j)]$. Because $suf_i$ and $suf_j$ are not the starting positions of any insert-head, it is true that $SA[ip(i-1)] = SA[ip(i)] - 1$ and $SA[ip(j-1)] = SA[ip(j)] - 1$. Therefore, $SA[ip(i-1)] = SA[ip(j-1)]$ and also $ip(i-1) = ip(j-1)$. Since $U_{i-1}, U_{j-1} \neq \varepsilon$ it follows that the first character of $U_{i-1}$ and $U_{j-1}$ is the same. ◀

While computing the *eCMS* of $\mathcal{C}$ w.r.t. $R$, we can simultaneously count how many suffixes fall in each insert-bucket. We recall that we can have at most $|R|$ insert-buckets, so the size of the array of counters called *bucket-counters* is $|R| \log N$ bits. By Lemma 5 we know that suffixes in the same insert-bucket have different preceding character only if one of them is an insert-head. By scanning again the collection, we just need to count how many suffixes belonging to the same insert-bucket come before each insert-head. In a sense, insert-heads work as run boundaries inside their insert-bucket, because they are preceded by a character that is different from the one preceding other non-insert-head suffixes. Therefore, we only need an additional counter for each insert-head to keep track of this quantity. We will store the counters in an array called *head-counters*. Inside a given insert-bucket we already know the total order of insert-heads, because we have sorted the whole set $K$ after the computation of *eCMS*, as mentioned in Section 3.2.

Since we are scanning $\mathcal{C}$ left-to-right, we know $MS[i]$ for every suffix, without the need of using predecessor queries as we explain next. By saving the *eCMS* in text order, we start from the first insert-head $k_1$. Every suffix $i$ before the starting position of $k_2$ have $MS[i] = (q_1 + (i - j_1), \ell_1 - (i - j_1))$, where $j_1 = 1$ is the starting position of $k_1$. Then, when

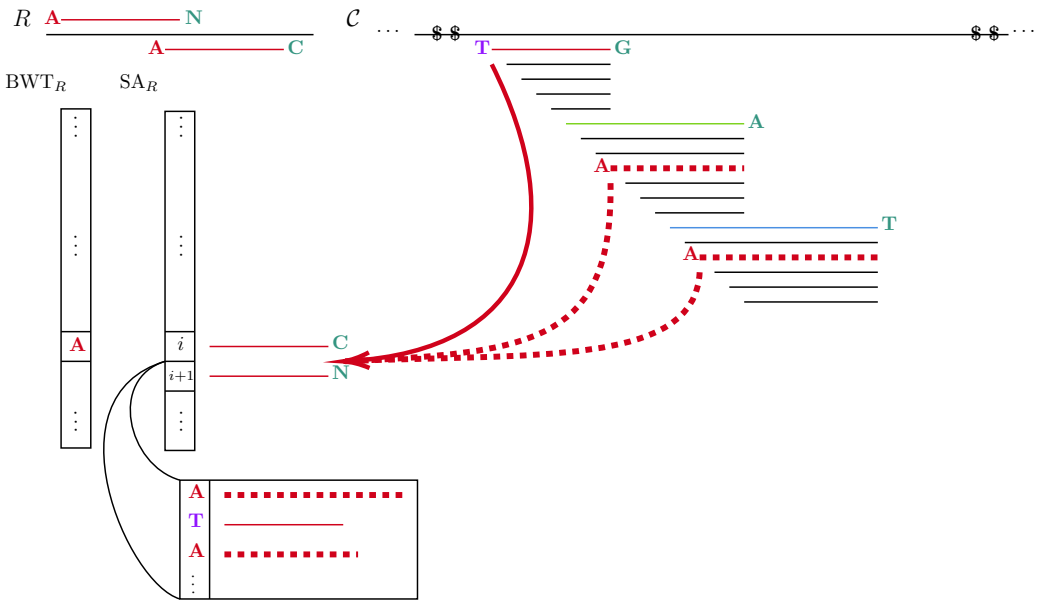


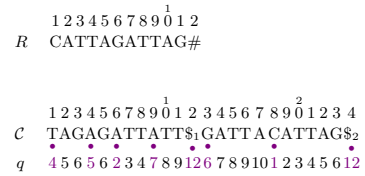

**Figure 2** Example showcasing the proposed heuristic. Solid coloured lines under $\mathcal{C}$ are matching factors for insert-heads, while solid black lines are matching factors for suffixes inbetween insert-heads. Dotted red lines are for suffixes inbetween insert-heads that share the same insert-point as the first solid red line (an insert-head). Suffixes inbetween insert-heads share the same preceding character (red A) with the reference $R$, while the red-coloured insert-head is preceded by a different character (purple T). Zooming in on the insert-bucket, we see that it can happen that the purple T goes between the red As, breaking what would have been a run of only red As.

we reach the starting position of $k_2$, we just have to repeat this procedure until every couple of insert-heads has been processed. We will compare suffix $i$ only with insert-heads stored in the corresponding insert-bucket, therefore we do not need to perform any predecessor query. Ultimately, the comparisons are made in constant time. Given $suf_i$, we can perform a binary search in the bucket corresponding to $ip(i)$ taking $\mathcal{O}(\log K_i)$ time, where $K_i$ is the set of insert-heads in the bucket with $ip(i)$. After finding the correct index using Lemma 4 and, if necessary, resorting to the rank of the sorted insert-heads, we increment the counter for that insert-head. The array of *head-counters* takes $\kappa \log N$ bits of space.

Building the BWT of $\mathcal{C}$ is then just a matter of interleaving bucket-counters and head-counters. For $1 \leq i \leq |R|$, let $x =$ bucket-counter$[i]$ be the number of suffixes in that bucket. If no insert-heads are present in the bucket, write $c = R[SA[i] - 1]$ in the output BWT$(\mathcal{C})$ $x$ times. Otherwise, if at least one insert-head is in the bucket, for each head-counter in the current insert-bucket write $c$ repeated as many times as indicated in the head-counter. Then, after the head-counter is processed, write the character that precedes the insert-head itself, namely $\mathcal{C}[j-1]$, where $j$ is the starting position of the insert-head. Each time we write a character from either the head-counter or the head itself, we subtract one from $x$. If at the end of this procedure, $x$ is not equal to 0, it means we still need to write that number of $c$ characters in the output BWT. This is because this amount of suffixes was bigger than any insert-head in their insert-bucket.

The main procedure is outlined in Algorithm 1 and the running time and space consumption are reported in Proposition 6. A full example can be found in Figure 3.

▶ **Proposition 6.** *Given $R$ and $\mathcal{C}$, we can compute BWT$(\mathcal{C})$ in $\mathcal{O}(N \log \kappa + N \log |R| + |R|)$ time and $\mathcal{O}(\kappa + |R|)$ space.*




| $i$ | $eCMS$ |
|---|---|
| 1 | ( 1, 4,4, G,S) |
| 2 | ( 4, 5,6, T,L) |
| 3 | ( 6, 2,4, T,L) |
| 4 | ( 9, 7,3,$\$_1$,S) |
| 5 | (12,12,0,$\$_1$,L) |
| 6 | (13, 6,5, C,S) |
| 7 | (18, 1,6,$\$_2$,S) |
| 8 | (24,12,0,$\$_2$,L) |

| $i$ | BWT$_R[i]$ | $SA_R[i]$ | bucket-counter[$i$] |
|---|---|---|---|
| 1 | G | 12 | 2 |
| 2 | T | 10 | 1 |
| 3 | T | 5 | 3 |
| 4 | G | 7 | 2 |
| 5 | C | 2 | 2 |
| 6 | # | 1 | 1 |
| 7 | A | 11 | 0 |
| 8 | A | 6 | 4 |
| 9 | T | 9 | 2 |
| 10 | T | 4 | 3 |
| 11 | A | 8 | 2 |
| 12 | A | 3 | 2 |

| $i$ | sorted $eCMS$ | head-counter[i] | prev. char. |
|---|---|---|---|
| 1 | (12,12,0,$\$_1$,L) | 0 | T |
| 2 | (24,12,0,$\$_2$,L) | 0 | G |
| 3 | ( 4, 5,6, T,L) | 2 | G |
| 4 | ( 9, 7,3,$\$_1$,S) | 0 | T |
| 5 | ( 6, 2,4, T,L) | 1 | G |
| 6 | (18, 1,6,$\$_2$,S) | 0 | A |
| 7 | (13, 6,5, C,S) | 2 | $\$_2$ |
| 8 | ( 1, 4,4, G,S) | 1 | $\$_1$ |

| $i$ | BWT$_\mathcal{C}[i]$ |
|---|---|
| 1 | T |
| 2 | G |
| 3 | T |
| 4 | T |
| 5 | T |
| 6 | G |
| 7 | T |
| 8 | G |
| 9 | C |
| 10 | G |
| 11 | A |
| 12 | A |
| 13 | A |
| 14 | $\$_2$ |
| 15 | A |
| 16 | T |
| 17 | T |
| 18 | T |
| 19 | $\$_1$ |
| 20 | T |
| 21 | A |
| 22 | A |
| 23 | A |
| 24 | A |

**Figure 3** Example of the construction of BWT($\mathcal{C}$). The colour purple is used to indicate a relationship with insert-heads, whereas red is used to indicate a relationship with $R$ and BWT$_R$. On the left, under $\mathcal{C}$ we mark insert-heads with a purple circle and highlight with the same colour $q_j$ when $j$ is an insert-head. In the middle part of the figure, entries of bucket-counter highlighted in red contains a positive number, meaning that no insert-head is contained within that bucket. On the other hand, entries coloured in purple tell us that we have at least one insert-head in that bucket. On the right, we show the full BWT of $\mathcal{C}$, where we use the same colour code. We also show with horizontal lines insert-buckets, highlighting how we interleaved information from bucket-counters and head-counters.

**Proof.** Computing all data structures for $R$ can be done in linear time and space in $|R|$. Computing the $eCMS$ of $\mathcal{C}$ takes $\mathcal{O}(N \log |R|)$ time and $\mathcal{O}(|R|)$ space using the approach described in Section 3.3. The computation of BWT($\mathcal{C}$) is bounded by the time of counting how many suffixes are smaller than each insert-head in a bucket with the same $ip(i)$. More specifically, $\sum_{1 \leq i \leq |R|} B_i \log K_i \leq |C| \log \kappa$, where $B_i$ is the set of suffixes belonging to insert-bucket $i$ and $K_i$ the set of insert-heads within the same insert-bucket $i$. The space consumption is dominated by the number of insert-heads and the size of the data structures on $R$. ◀

Because we are working with highly similar strings, we expect to have few insert-heads, having long matches between any string of $\mathcal{C}$ and $R$. This makes the sorting part of insert-heads very fast in practice, due to $\kappa$ being small. Also, the process of binary searching is conducted bucket by bucket, so the number of heads in the same bucket is expected to be smaller than $\kappa$.

Moreover, if the insert-heads are concentrated in a few insert-buckets we can entirely skip the computation for each bucket without insert-heads. More information on real-life datasets related to this insight can be found in Section 6.2.

## 5 Implementation details

The algorithm starts by first augmenting the reference with characters that occur in $\mathcal{C}$ but not in $R$ so that we have a well-defined insert point for each suffix of the collection.

**Algorithm 1** CMS-BWT.

**Input:** reference $R$, collection $\mathcal{C}$
**Output:** BWT($\mathcal{C}$)

compute $SA_R$, $ISA_R$, $LCP_R$, $PSV - NSV(LCP_R)$
bucket-counters $\leftarrow [0] * N$
$eCMS \leftarrow [\,]$
$p_{\text{prev}} \leftarrow -\infty$
**for** $i \leftarrow 1$ ***to*** $N$ **do**
  $\langle i, p_i, \ell_i, c_i, x_i \rangle \leftarrow$ computeMS($R, \mathcal{C}, i$)
  **if** $p_i \neq p_{prev} + 1$ **then**
    $eCMS$.add($\langle i, p_i, \ell_i, c_i, x_i \rangle$)
    mark bucket-counter[$p_i$]  `// insert-bucket has an insert-head`
    bucket-counter[$p_i$] $++$
    $p_{\text{prev}} \leftarrow p_i$
  **else**
    bucket-counter[$p_i$] $++$
  **end**
**end**
sort $eCMS$
head-counters $\leftarrow [0] * K$
**for** $i \leftarrow 1$ ***to*** $N$ **do**
  **if** *bucket-counters*[$p_i$] *is marked* **then**
    $j \leftarrow$ binary-search correct position of $suf_i$ in $K_{p_i}$
    head-counters[$j$] $++$
  **end**
**end**
**for** $i \leftarrow 1$ ***to*** $|R|$ **do**
  $x \leftarrow$ bucket-counters[$i$]
  **if** $i > 1$ **then** $char \leftarrow R[SA_R[i-1]]$
  **else** $char \leftarrow R[|R|]$
  **if** *bucket-counter*[$i$] *is marked* **then**
    **for** $j \in indices(K_i)$ **do**  `// set of indices of` $K_i$
      write $char$ head-counters[$j$] times
      write character preceding $j$th head
      $x = x -$ head-counters[$j$] $- 1$
    **end**
    **if** $x > 0$ **then**
      write $char$ $x$ times
    **end**
  **else**
    write $char$ bucket-counter[$i$] times
  **end**
**end**

Then, we use `libsais` [16] to build $SA_R$ and $LCP_R$. We chose this tool because it has been experimentally proven to be one of the fastest tools for general-purpose suffix array construction. For the $LCP$ array it uses the $\Phi$ method [17]. The data structure for *PSV-NSV* queries on the $LCP_R$ is based on the work of Cánovas and Navarro [7].

For sorting the *eCMS*, we first rename each insert-head with a metacharacter based on the rank of the partial lexicographic order of the substrings associated with each insert-head. Then, by rearranging these metacharacters in text order we use again `libsais` to compute the suffix array of this metacharacter string.

When profiling the implementation, we found that the number of distinct insert-heads, i.e. the number of different tuples in $K$, grows even slower than the total set. For example, looking at the dataset consisting of 333 copies of Human Chromosome 19 described in Section 6.2, we have only 4,355,600 unique insert-heads versus $\kappa = 174,532,868$. Moreover, when performing binary search comparisons, more than 60% of the total number of comparisons were resolved by comparing the length plus $x_i$ information stored in the *eCMS*. Combining these two insights led us to another heuristic based on a two-layered binary search. First, we compare the length information $\ell_i = \ell_{k_i} - (i - j)$ along with $x_{k_i}$ of a suffix $i$ with insert-head $pred_K(i) = k_i$ starting at position $j$ with the set of unique insert-heads of its insert-bucket. Then, if the pair $\ell_i$ and $x_{k_i}$ is different from any other insert-head we increment the counter for the insert-head pointed to by this first binary search. Otherwise, we have to refine the search by comparing $suf_i$ with the whole set of insert-heads having $ip(i)$, $\ell = \ell_i$ and $x_{k_i}$. This technique led to a speedup in the binary search phase of between 10% and 20%.

To avoid continuous cache misses due to loading different subsets of insert-heads with different insert points during binary searching, we put a number of suffixes in a buffer divided into insert-buckets. After the buffer is at its full capacity, we proceed to process in bulk suffixes in the same insert-bucket, easing the loading in cache of subsets of insert-heads. For all of our experiments, we set this buffer to 2GB, but it can be arbitrarily chosen by the user.

Lastly, we also implemented a variant of `CMS-BWT` trading off space for running time. This was achieved by writing to disk some of the data structures involved in different phases of the algorithm. This version saves roughly a third of the space used by the non-memory-saving implementation.

## 6 Experiments

We implemented our algorithm for computing the BWT in C++. Our implementation, `CMS-BWT`, is available at `https://github.com/fmasillo/CMS-BWT`. The experiments were conducted on a desktop equipped with 64GB of RAM DDR4-3200MHz and an Intel(R) Core(R) i9-11900 @ 2.50GHz (with turbo speed @ 5GHz) with 16 MB of cache. The operating system was Ubuntu 22.04 LTS, the compiler used was `g++` version 11.3.0 with options `-std=c++20 -O3 -funroll-loops -march=native` enabled.

### 6.1 Tools compared

We compared two different implementations of `CMS-BWT` (simple and memory-saving) to the following four tools:

1. `big-BWT` [5], a tool which computes both the BWT and the suffix array. It is specifically made for highly repetitive data. We used the default parameters (`-w = 10`, `-p = 100`) and the `-f` flag to parse fasta files as input. We chose the default parameters in order to be consistent with the literature [5, 3, 4]. This tool outputs the entire BWT.

To double-check the choice of parameters, we performed further experiments on each combination of $w \in \{6, 8, 10\}$ and $p \in \{50, 100, 200, 400, 800\}$ as reported in [5]. On the basis of these experiments, we concluded that the default parameters $w = 10$ and $p = 100$ lead to the best combination of memory and time for the two datasets evaluated. It should be noted that the variation in memory and time between the different parameters is non-negligible, reaching a 7 times larger peak memory consumption and a 40% slowdown in running time for the `sars-cov2` dataset (data not shown).

2. `r-pfBWT` [26], is a tool improving on plain PFP. It has been shown to be both faster and to use less space than `big-BWT` for big enough dataset sizes. We run the experiments using `--bwt-only --w1 10 --w2 5` as flags. This tool outputs the run-length encoded BWT.
3. `grlBWT` [10], a tool computing the BCR BWT [2] again with a focus on highly repetitive data. We used the default parameters. This tool outputs the run-length encoded BWT.
4. `ropeBWT2` [19], a highly optimized tool to compute the BCR BWT on DNA data. We used the flag `-R` to skip the reverse complement. We also compare the effect of adding the `-P` flag, which limits the software to execute in single-threaded mode at all times. This tool outputs the entire BWT.

## 6.2 Datasets

In our experiments, we used two publicly available datasets. The first dataset, called `chr19` contains copies of the Human Chromosome 19 from the 1000 Genomes Project [28]. The second dataset, named `sars-cov2`, consists of copies of SARS-CoV2 genomes taken from COVID-19 Data Portal [1]. Some additional metadata can be found in Table 1.

The total size of both datasets is 60GB. We took increasing prefixes of size 1GB, 10GB, 20GB, 40GB, and 60GB. For the explicit number of sequences contained in each dataset see Table 2.

For example, looking at 20GB of `chr19` data, where we have around 333 copies of Human Chromosome 19, we have insert-heads only in 6% of the buckets. This means that around 94% of the suffixes will not be compared against any insert-head, speeding up the whole process.

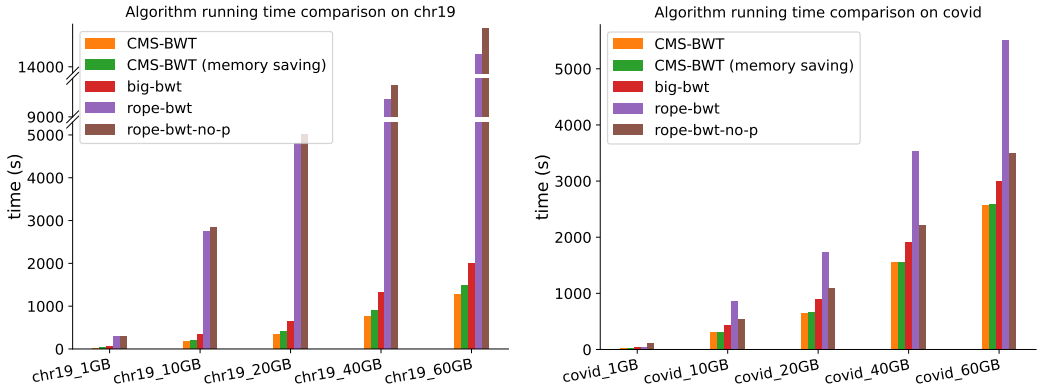
**Table 1** Datasets used in experiments. In column 3, we specify the alphabet size $\sigma$, in column 4 the number $r$ of runs of the BWT, in column 5 the number of insert-heads, and in column 6 the number of unique insert-heads. In our experiments, we use prefixes of each dataset up to 60GB. The last three columns refer to the 20GB prefix.

| Name | Description | $\sigma$ | $r$ (20 GB) | no. of i-heads (20 GB) | no. unique i-heads (20 GB) |
|---|---|---|---|---|---|
| `chr19` | Human Chromosome 19 | 5 | 36 723 404 | 174 532 868 | 4 355 600 |
| `sars-cov2` | SARS-CoV2 genome | 14 | 19 075 277 | 253 188 521 | 1 466 183 |

[1] We used the following command to download in bulk the data using the CDP File Downloader: `java -jar cdp-file-downloader.jar - -domain=VIRAL_SEQUENCES - -datatype=SEQUENCES - -format=FASTA - -location=/home/data/ - -email=xxx@xxx.xx - -protocol=FTP`

**Table 2** Number of sequences present in each dataset.

| Name | 1GB | 10GB | 20GB | 40GB | 60GB |
|---|---|---|---|---|---|
| `chr19` | 17 | 167 | 333 | 666 | 1 000 |
| `sars-cov2` | 36 204 | 332 209 | 659 441 | 1 312 058 | 1 966 237 |




**(a)** Running time comparison on different subsets of copies of Chromosome 19.

**(b)** Running time comparison on different subsets of copies of SARS-CoV2 genomes.

**Figure 4** Running time comparison of tools outputting the full BWT.

## 6.3 Results

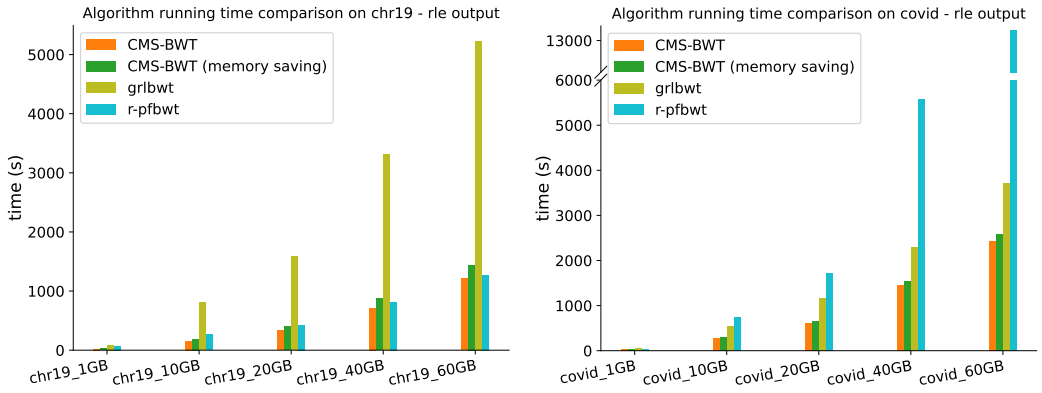
As already pointed out in Section 6.1, the output of the tools can either be the full BWT or the run-length encoded BWT. This can be a non-negligible time overhead. Therefore, when comparing to tools that output the whole BWT we will also write this version of the BWT to disk. On the other hand, when comparing `CMS-BWT` to `r-pfBWT` and `grlBWT` we will write to disk the run-length encoded BWT.

In Figures 4a, 4b, 5a and 5b, we report the comparison of the running time of the five tools divided by dataset and output type.

On the `chr19` dataset, we are always the fastest tool compared to other tools that output the uncompressed BWT. More specifically, comparing the non-memory-saving implementation at 60GB of data, we are 57% faster than `big-BWT` and 10 times faster than `ropeBWT2` with and without `-P`. Compared to the tools that output the run-length encoded BWT, our fastest implementation is always the winner, while at 60GB, our memory-saving implementation takes 12% more time than `r-pfBWT`. Both implementations outperform `grlBWT`, e.g. at 60GB of data they take a fourth of the time.

On the `sars-cov2` dataset we are always the fastest tool in both settings. For example, at 60GB of data, we are faster than: `big-BWT` by 17%, `ropeBWT2` with `-P` by 114%, `ropeBWT2` with no `-P` by 35%, `r-pfBWT` by 445% and `grlBWT` by 53%.

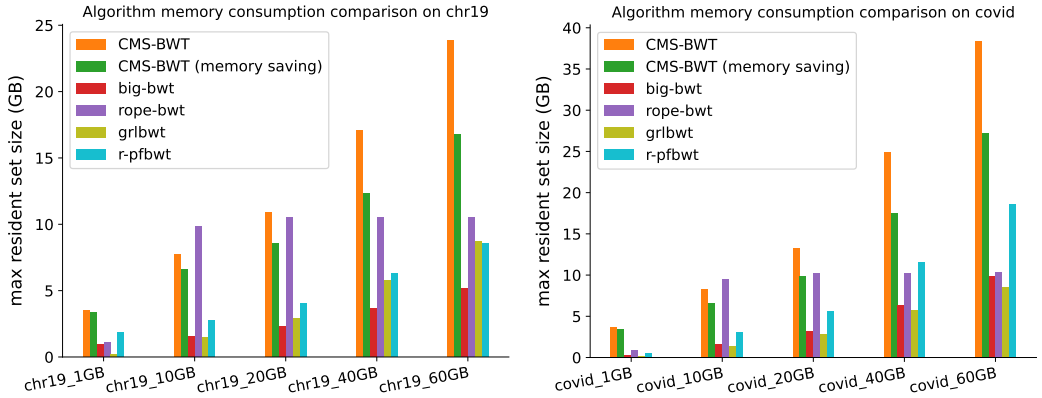
In Figure 6a and 6b we show the memory footprint of the five tools. As one can notice, our tool has the highest memory requirement. However, it can be noted that the memory-saving variant of `CMS-BWT` on bigger sizes of both datasets requires always less than half of the input size in space. On the `sars-cov2` dataset we have a higher memory footprint than on `chr19` because for the same size of the datasets we have a significantly higher number of strings in the collection, leading to more insert-heads.




**(a)** Running time comparison on different subsets of copies of Chromosome 19.

**(b)** Running time comparison on different subsets of copies of SARS-CoV2 genomes.

**Figure 5** Running time comparison of tools outputting the run-length encoded BWT.




**(a)** Comparison of tools on different subsets of copies of Chromosome 19.

**(b)** Comparison of tools on different subsets of SARS-CoV2 genomes.

**Figure 6** Peak memory measured as maximum resident set size in GB.

## 7 Conclusions

We presented a new algorithm for constructing the BWT of a collection of highly similar strings in compressed space. An experimental evaluation of two different implementations shows that our algorithm is competitive with state-of-the-art tools. Most of the time, both implementations outperform the other tools in terms of running time, but they are also the heaviest w.r.t. space consumption.

Future work will focus on parallelizing the implementation to allow taking advantage of multicore CPUs that are widespread nowadays. It is fairly straightforward to assign distinct sequences to a pool of multiple threads to compute the matching statistics. Another phase that would directly benefit from multi-threading is the for-loop at line 18 in Algorithm 1. With careful handling of locks for each head-counter, this is easily parallelizable, dividing the for-loop into equal parts. Moreover, we are going to investigate other ways of reducing memory consumption to close the gap between `CMS-BWT` and competing tools.

We are also working on extending our algorithm to incorporate the computation of $SA$-samples. This will allow us to build the $r$-index. With careful implementation, our tool can be extended to compute $SA$-samples along with bucket- and head-counters, without changing either the time or space bounds given in Proposition 6.

## References


1 Uwe Baier. Linear-time suffix sorting - A new approach for suffix array construction. In *Proc. of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, volume 54 of *LIPIcs*, pages 23:1–23:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

2 Markus J. Bauer, Anthony J. Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theor. Comput. Sci.*, 483:134–148, 2013.

3 Christina Boucher, Davide Cenzato, Zsuzsanna Lipták, Massimiliano Rossi, and Marinella Sciortino. r-indexing the eBWT. In *Proc. of the 28th International Symposium on String Processing and Information Retrieval (SPIRE 2021)*, volume 12944 of *Lecture Notes in Computer Science*, pages 3–12. Springer, 2021.

4 Christina Boucher, Ondrej Cvacho, Travis Gagie, Jan Holub, Giovanni Manzini, Gonzalo Navarro, and Massimiliano Rossi. PFP compressed suffix trees. In *Proc. of the Symposium on Algorithm Engineering and Experiments (ALENEX 2021)*, pages 60–72. SIAM, 2021.

5 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms Mol. Biol.*, 14(1):13:1–13:15, 2019.

6 Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, DIGITAL System Research Center, 1994.

7 Rodrigo Cánovas and Gonzalo Navarro. Practical compressed suffix trees. In *Proc. of the 9th International Symposium Experimental Algorithms, SEA 2010*, volume 6049 of *LNCS*, pages 94–105. Springer, 2010.

8 William I. Chang and Eugene L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.

9 Fabio Cunial, Olgert Denas, and Djamal Belazzougui. Fast and compact matching statistics analytics. *Bioinform.*, 38(7):1838–1845, 2022.

10 Diego Díaz-Domínguez and Gonzalo Navarro. Efficient construction of the BWT for repetitive text using string compression. In *Proc. of 33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)*, volume 223 of *LIPIcs*, pages 29:1–29:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

11 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proc. of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 390–398. IEEE Computer Society, 2000.

12 Johannes Fischer. Combined data structure for previous- and next-smaller-values. *Theor. Comput. Sci.*, 412(22):2451–2456, 2011.

13 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):2:1–2:54, 2020.

14 Sara Giuliani, Giuseppe Romana, and Massimiliano Rossi. Computing maximal unique matches with the $r$-index. In *Proc. of the 20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

15 Keisuke Goto. Optimal time and space construction of suffix arrays and LCP arrays for integer alphabets. In *Proc. of the Prague Stringology Conference 2019*, pages 111–125. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2019.

16 Ilya Grebnov. Code for libsais. URL: `https://github.com/IlyaGrebnov/libsais`.

**17** Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. Permuted longest-common-prefix array. In *Proc. of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009)*, volume 5577 of *LNCS*, pages 181–192. Springer, 2009.

**18** Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4):357–359, 2012.

**19** Heng Li. Fast construction of FM-index for long sequence reads. *Bioinform.*, 30(22):3274–3275, 2014.

**20** Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinform.*, 26(5):589–595, 2010.

**21** Zhize Li, Jian Li, and Hongwei Huo. Optimal in-place suffix sorting. *Inf. Comput.*, 285(Part):104818, 2022.

**22** Zsuzsanna Lipták, Francesco Masillo, and Simon J. Puglisi. Suffix sorting via matching statistics. In *Proc. of the 22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

**23** Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy, Rachel Coston, Steven L. Salzberg, and Aleksey V. Zimin. Mummer4: A fast and versatile genome alignment system. *PLoS Comput. Biol.*, 14(1), 2018.

**24** Ge Nong. Practical linear-time $O(1)$-workspace suffix sorting for constant alphabets. *ACM Trans. Inf. Syst.*, 31(3):15, 2013.

**25** Ge Nong, Sen Zhang, and Wai Hong Chan. Two efficient algorithms for linear time suffix array construction. *IEEE Trans. Computers*, 60(10):1471–1484, 2011.

**26** Marco Oliva, Travis Gagie, and Christina Boucher. Recursive prefix-free parsing for building big BWTs. *bioRxiv*, pages 2023–01, 2023.

**27** Massimiliano Rossi, Marco Oliva, Ben Langmead, Travis Gagie, and Christina Boucher. MONI: A pangenomic index for finding maximal exact matches. *J. Comput. Biol.*, 29(2):169–187, 2022.

**28** The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015.

**29** Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983.

# Approximation Schemes for Min-Sum $k$-Clustering

**Ismail Naderi**
Department of Computer Science, University of Alberta, Edmonton, Canada

**Mohsen Rezapour**
Department of Computer Science, University of Alberta, Edmonton, Canada
Department of Computer Science and Statistics, K. N. Toosi University of Technology, Tehran, Iran

**Mohammad R. Salavatipour**
Department of Computer Science, University of Alberta, Edmonton, Canada

## Abstract

We consider the Min-Sum $k$-Clustering ($k$-MSC) problem. Given a set of points in a metric which is represented by an edge-weighted graph $G = (V, E)$ and a parameter $k$, the goal is to partition the points $V$ into $k$ clusters such that the sum of distances between all pairs of the points within the same cluster is minimized.

The $k$-MSC problem is known to be APX-hard on general metrics. The best known approximation algorithms for the problem obtained by Behsaz, Friggstad, Salavatipour and Sivakumar [Algorithmica 2019] achieve an approximation ratio of $O(\log |V|)$ in polynomial time for general metrics and an approximation ratio $2 + \epsilon$ in quasi-polynomial time for metrics with bounded doubling dimension. No approximation schemes for $k$-MSC (when $k$ is part of the input) is known for any non-trivial metrics prior to our work. In fact, most of the previous works rely on the simple fact that there is a 2-approximate reduction from $k$-MSC to the balanced $k$-median problem and design approximation algorithms for the latter to obtain an approximation for $k$-MSC.

In this paper, we obtain the first Quasi-Polynomial Time Approximation Schemes (QPTAS) for the problem on metrics induced by graphs of bounded treewidth, graphs of bounded highway dimension, graphs of bounded doubling dimensions (including fixed dimensional Euclidean metrics), and planar and minor-free graphs. We bypass the barrier of 2 for $k$-MSC by introducing a new clustering problem, which we call min-hub clustering, which is a generalization of balanced $k$-median and is a trade off between center-based clustering problems (such as balanced $k$-median) and pair-wise clustering (such as Min-Sum $k$-clustering). We then show how one can find approximation schemes for Min-hub clustering on certain classes of metrics.

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

**Keywords and phrases** Approximation Algorithms, Clustering, Dynamic Programming

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.84

**Funding** *Mohammad R. Salavatipour*: This research project was supported by the last authors NSERC Discovery grant.

## 1 Introduction

Clustering is a fundamental problem in many areas of data analysis and machine learning and has many applications across various fields. Given a set of points with a notion of similarity (distance) between every pair of points, in a typical $k$ clustering problem, the task is to partition the points into $k$ clusters to minimize dissimilarities of the points that fall into the same cluster.

In the well-known *center-based* $k$-clustering problems (such as $k$-*center*, $k$-*median*, $k$-*means*), the partition is obtained by selecting a set of $k$ *centers* and assigning each point to its nearest center. The clusters are then *evaluated* based on the distances between the points and their centers: in the case of $k$-center, the objective is to minimize the maximum distance of a point to its nearest center, while in the case of $k$-median ($k$-means), respectively, the


(a) (b)

**Figure 1** Clustering of a set of points: (a) a possible center-based clustering induced by a Voronoi diagram of two cluster centers , and (b) a min-sum $k$-clustering solution for $k = 2$. Observe that the min-sum $k$-clustering solution in (b) places all outliers into a separate cluster.

objective is to minimize the sum of distances (the sum of squared distances, respectively) between points and their centers. Compared to other clustering algorithms, center-based algorithms are efficient for clustering large datasets as the main task reduces to selecting $k$ centers; once we decided on the set of centers, points that are closest to a particular center are considered to be part of the cluster represented by that center. Center-based clustering algorithms are not always precise because they heavily rely on the assumption that each cluster has a *spherical* shape and hence can be represented by one center.

In *pair-wise* $k$-clustering, on the other hand, the goal of partitioning is to minimize the dissimilarity between pairs of points that are in the same cluster. For example, in the case of the *$k$-diameter* problem, the goal is to minimize the maximum distance between any two points in a cluster; or in the *min-sum $k$-clustering* problem, the goal is to minimize the sum of distances between all pairs of the points within the same cluster.

Unlike center-based clustering problems, min-sum $k$-clustering (which is the main focus of this paper) is less sensitive to the shape of clusters because it forms clusters based on the pair-wise distances between points rather than the distances of points to their cluster center. Also, as observed in [8], min-sum $k$-clustering can handle (detect) noises (outliers) in an effective way: in scenarios where data include well-defined clusters and a limited number of scattered noises (outliers), assigning an outlier to one of the clusters would be more costly than placing it in an outlier cluster that holds all the outliers. This results in a solution with a separate cluster specifically for outliers, avoiding the limitations of center-based clustering algorithms, which rely on Voronoi partitioning to divide the data space into clusters and are unable to handle overlapping cluster spaces. See Figure 1.

We now formally define the min-sum $k$-clustering problem. Given a metric space over a set of $n$ points $V$ with metric distances $d(u, v)$ between any two $u, v \in V$. We assume the metric is induced by an edge-weighted graph $G = (V, E)$. In the **Min-Sum $k$-Clustering** problem (**$k$-MSC**), the goal is to partition points $V$ into $k$ clusters $C_1, ..., C_k$ to minimize the sum of pairwise distances between points assigned to the same cluster: $\sum_{i=1}^{k} \sum_{\{u,v\} \subseteq C_i} d(u, v)$. This problem is closely related to the **Balanced $k$-Median** problem (**$k$-BM**), with the same input as in **$k$-MSC**. Here, the goal is to select $k$ points $c_1, ..., c_k \in V$ as the centers of the clusters and partition points $V$ into clusters $C_1, ..., C_k$ to minimize $\sum_{i=1}^{k} |C_i| \sum_{v \in C_i} d(v, c_i)$.

## Related Works

Sahni and Gonzalez introduced $k$-MSC in 1976 [13]. They showed the problem is $NP$-hard and provided a polynomial time $k$-approximation algorithm for the $k$-Max Cut problem, which is the dual of **$k$-MSC** and involves partitioning points into $k$ clusters to maximize the distance between points in different clusters. Kann et al. [12] showed it is $NP$-Hard

to approximate non-metric ***k*-MSC** within $O(n^{2-\epsilon})$ for any $\epsilon > 0$ and $k > 3$. Later, Cohen-Addad et al. [6] proved that it is $NP$-hard to approximate metric ***k*-MSC** within 1.415.

Guttman-Beck and Hassin [11] showed that ***k*-BM** and ***k*-MSC** are closely related. They showed an algorithm with $\rho$ approximation for one of these problems implies a $2\rho$ approximation for the other. In the literature, most of the previous work (with a guaranteed approximation factor) for ***k*-MSC** make use of this reduction. Guttman-Beck and Hassin [11] showed that **k-BM** can be solved in time $n^{O(k)}$ by guessing the cluster centers and sizes and finding the minimum-cost assignment from clients to these centers. This results in a 2-approximation solution for the min-sum $k$-clustering problem when $k$ is fixed. Bartal et al. [3] introduced the first polynomial time approximation algorithm for both ***k*-MSC** and ***k*-BM** in metric spaces. They devised an algorithm with an approximation factor of $O(\frac{1}{\epsilon}\log^{1+\epsilon} n)$ and running time of $n^{\frac{1}{\epsilon}}$ for ***k*-BM**. The algorithm is based on the embedding of metric spaces into hierarchically separated trees (HSTs). They also provided a bi-criteria approximation algorithm with a constant approximation factor with $O(k)$ clusters. Later, Behsaz et al. [4] improved the result by utilizing the properties of HSTs through a direct dynamic programming approach, leading to a $O(\log n)$ approximation algorithm for both ***k*-MSC** and ***k*-BM**. This is the current best result for general metrics. They also present a quasi-polynomial time approximation scheme for ***k*-BM** in metrics with constant doubling dimensions, leading to a $(2+\epsilon)$-approximation algorithm for the min-sum $k$-clustering problem that runs in quasi-polynomial time. More recently Banerjee et al. [2] gave a bicriteria approximation for ***k*-MSC** with outliers: for any $\epsilon > 0$, given an instance with $n$ points and any integer $n' \leq n$, their algorithm finds a solution that clusters at least $(1-\epsilon)n'$ points whose cost is poly$(1/\epsilon)$ times the optimum clustering of $n'$ points.

For small values of $k$, Vega et al. [9] introduced the first polynomial time approximation scheme for ***k*-MSC** in metric spaces. The running time of their algorithm is $O(n^{3k}2^{\epsilon^{-k^2}})$. Czumaj and Sohler [8], presented a $(4+\epsilon)$ approximation algorithm for ***k*-MSC** in metric spaces with a running time of linear for $k = o(\log n/\log\log n)$.

## Our Results and Techniques

As mentioned earlier, the previous methods for designing approximation for ***k*-MSC** attempt to approximate the cost using a center-based clustering objective (such as ***k*-BM** [3, 4] or a capacitated version of $k$-median [2]). Such methods have a barrier of 2 (even for tree metrics). A key challenge in extending the framework of [4] to work directly for ***k*-MSC** is to develop a compact representation of the cluster types in a near-optimal solution that can capture the essence of the cluster without relying on a center.

Here we introduce a new clustering objective that is in between the pair-wise distances objective of ***k*-MSC** and the center-based objective of ***k*-BM**, which we call min-hub clustering. We show that for metrics with a nice hierarchical decomposition (such as graphs of bounded treewidth, or bounded doubling dimension), the objective of min-hub clustering is a good (namely $(1+\epsilon)$) approximation of ***k*-MSC** and how one can obtain an approximation scheme for the new objective (and hence one for ***k*-MSC**).

In center-based clustering, a cluster is represented by a single center. However, as demonstrated in Figure 1 (see the outlier cluster in red), not all ***k*-MSC** clusters can be represented by a single center. To address this, we explore the possibility of using multiple centers to represent a cluster. Our results show that a cluster in the ***k*-MSC** solution can be represented by $O_\epsilon(1)$ centers, which we refer to as *hubs*, while incurring an error of $(1+\epsilon)$. Specifically, let $H$ be a set of hubs. The *hub-distance* between two points $u$ and $v$ in a cluster

$C$ is defined as the shortest path between the points that passes through hub points in $H$. Our results show that there exists a set of $H$ of constant size (depending on $\epsilon$) such that the sum of distances between all pairs of points within $C$ is "almost" equal to the sum of hub-distances between pairs of points in $C$. This suggests that the network interconnecting the hubs, called the *backbone structure*, carries the majority of the connection flow in the cluster. We represent a cluster by the type of its backbone structure and the distribution of points around its hubs.

In Section 2, we consider the special case of tree metrics. We construct a dynamic program for ***k*-MSC** on tree metrics that have a logarithmic height. In Section 3, we extend our approach to cover metrics with bounded treewidth, thereby covering general trees as well.

▶ **Theorem 1.** *There is a quasi-polynomial time algorithm that, given an instance of* ***k*-MSC** *on a metric of treewidth $f$, for any $\epsilon > 0$ finds a $(1+\epsilon)$-approximate solution in time $n^{O(f^2+(\frac{\log n}{\epsilon})^{\sigma+1})}$, where $\sigma$ depends on $\epsilon$.*

It is worth pointing out that, if one tries to extend the result from trees to graphs with treewidth $f$ in a natural way, the algorithm will have a run time of the form $n^{(\frac{\log n}{\epsilon})^{f^2+\sigma+1}}$ (instead of $n^{O(f^2+(\frac{\log n}{\epsilon})^{\sigma+1})}$), which is still quasi-polynomial for fixed $f$, but will not be quasi-polynomial if $f = \text{Polylog}(n)$. This is essential to obtain the next three theorems, as we use embeddings into graphs with treewidths $f = \text{Polylog}(n)$.

In Section 4, using frameworks from [14], [10], and [7], we expand our results to three additional metric classes: bounded doubling metrics, bounded highway dimension metrics, and minor-free metrics, respectively.

▶ **Theorem 2.** *There is a quasi-polynomial time algorithm that, given an instance of* ***k*-MSC** *on a metric of doubling dimension $D$, for any $\epsilon > 0$ finds a $(1+\epsilon)$ approximate solution in time $n^{O((\frac{D\log n}{\epsilon})^{2D}+(\frac{\log n}{\epsilon})^{\sigma+1})}$.*

▶ **Theorem 3.** *There is a quasi-polynomial time algorithm that, given an instance of* ***k*-MSC** *on a metric highway dimension $D$ and violation $\lambda$, for any $\epsilon > 0$ finds a $(1+\epsilon)$ approximate solution in time $n^{O((\log n)^{\alpha}+(\frac{\log n}{\epsilon})^{\sigma+1}))}$, where $\alpha = O(\log^2(\frac{D}{\epsilon\lambda})/\lambda)$.*

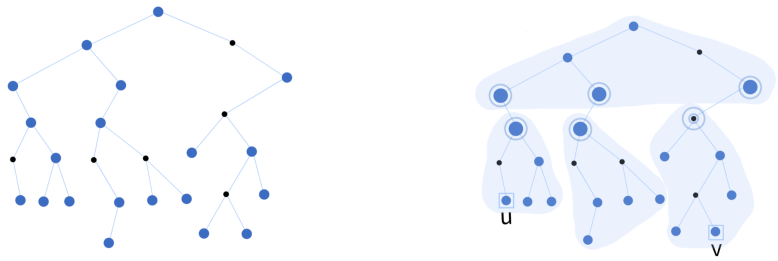
▶ **Theorem 4.** *There is a quasi-polynomial time algorithm that, given an instance of* ***k*-MSC** *in minor-free metrics, for any $1/2 > \epsilon > 0$ finds a $(1+\epsilon)$ approximate solution in time $n^{\epsilon^{-O(1)}\log^{O(1)} n}$.*

## 2 The $k$-MSC Problem in Tree Metrics

In this section, we construct a dynamic program for ***k*-MSC** on trees. Consider metric $(V, d)$ induced by an edge-weighted tree $T = (V, E)$. Let $w(e)$ denote the weight of edge $e$ in $E$.

We let $T$ be rooted at an arbitrary vertex $r \in V$. The parent of a vertex $v \in V \setminus \{r\}$ is the vertex adjacent to $v$ on the path from $v$ to $r$. If $u$ is the parent of $v$ then $v$ is a *child* of $u$. A tree vertex is called a *leaf* if it has no children and is called an *internal* vertex otherwise. The *level* of each node is the number of edges on the path from it to $r$. The *height* of the tree is the level of the leaf node with the highest level. We use $T_v$ to denote the *subtree* rooted at $v$, $V(T_v)$ and $E(T_v)$ to denote the vertex set and the edge set of $T_v$, respectively. By introducing zero-weight edges and nodes, we convert the tree into an equivalent binary tree. Note that the resulting binary tree has at most $2|V|$ nodes.

We use $C \subseteq V$ to denote a cluster and $D(C)$ to denote the total sum of the distances between all pairs of points in $C$; i.e., $D(C) = \sum_{\{u,v\}\subseteq C} d(u, v)$. We use $H \subseteq V$ to indicate a set of points referred to as *hubs*. The distance between any two points $u$ and $v$ in $C$, when




(a) (b)

**Figure 2** (a) A cluster on the tree, where the blue circles specify points of this cluster. (b) Shaded regions highlight the resulting groups by applying Lemma 5 on the cluster. Notice that the distance between any two points of the cluster that belong to different groups (such as $u$ and $v$) is equal to their hub-distance, $d_H(u,v)$, as long as $H$ contains the border nodes of the groups. The larger circles around the nodes depict the border nodes of the groups (so the proper hubs of the cluster).

**Algorithm 1** Tree Partitioning Algorithm.

$\mathbb{C}_\nu \leftarrow \emptyset$
$\eta \leftarrow \max\{\nu|C|, 1\}$
$L \leftarrow \{v \in V : \quad \frac{1}{2}\eta \le |V(T_v) \cap C| \le \eta\}$
**while** $L \neq \emptyset$ **do**
  $\hat{v} \leftarrow v \in L$ ▷ If multiple, select $v$ with the lowest level.
  $g \leftarrow V(T_{\hat{v}})$
  $\mathbb{C}_\nu \leftarrow \mathbb{C}_\nu \cup \{g\}$
  remove $T_{\hat{v}}$ from $T$
  $L \leftarrow \{v \in V(T) : \quad \frac{1}{2}\eta \le |V(T_v) \cap C| \le \eta\}$
**end**
$\mathbb{C}_\nu \leftarrow \mathbb{C}_\nu \cup \{V(T_r)\}$

measured through hubs in $H$, is called the *hub-distance* and is denoted by $d_H(u,v)$. This is the length of the shortest path between the two points that goes through hub points in $H$; i.e., $d_H(u,v) = \min_{h_1,h_2 \in H} \big(d(u,h_1) + d(h_1,h_2) + d(h_2,v)\big)$. Let $p_H(u,v)$ represent the path between points $u$ and $v$ that passes through hub points in $H$ and has the length of $d_H(u,v)$. The *sum of pairwise hub-distances* for the points in $C$ is represented by $D_H(C)$ and is equal to the total sum of the hub-distances between all pairs of points in $C$; i.e., $D_H(C) = \sum_{\{u,v\} \subseteq C} d_H(u,v)$. Note that $D_V(C)$=$D(C)$.

The following lemma shows how to find a (constant-size) set of hubs that represents a given cluster in metrics induced by a tree metric. See Figure 2. For a subset of nodes $g \subseteq V$, we use $\delta(g) = \{v \in g : \ uv \in E \ \& \ u \notin g\}$ to denote the *border nodes* of $g$.

▶ **Lemma 5.** *Let $C \subseteq V$ be a cluster and let $T = (V,E)$ be a given binary tree. For any $\nu > 0$, there exists a partition of $V$ into a set of groups $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ such that all of the following properties hold: (i) the subgraph induced by each group $g \in \mathbb{C}_\nu$ is connected. (ii) for each group $g \in \mathbb{C}_\nu$, $|g \cap C| \in [1, \max\{1, \nu|C|\}]$. (iii) $|\mathbb{C}_\nu| = O(1/\nu)$. (iv) $\forall g \in \mathbb{C}_\nu, \quad |\delta(g)| = O(1/\nu)$.*

**Proof.** We use Algorithm 1 to compute $\mathbb{C}_\nu$. The algorithm iteratively selects a subtree $T_{\hat{v}}$, with approximately $\frac{\nu}{2}$ of the total number of points $|C|$, adds the vertex set $V(T_{\hat{v}})$ to $\mathbb{C}_\nu$, and removes $T_{\hat{v}}$ from $T$. The number of iterations (i.e. the number of groups made by the algorithm) is at most $2/\nu$, and every vertex of $V$ belongs to one group.

Note that there is at most one edge between any two groups, so $|\delta(g)| = O(1/\nu)$, $\forall g \in \mathbb{C}_\nu$. The subgraphs induced by $g_i$'s are connected by construction. Thus, the algorithm has constructed a partition with the desired properties, as shown in Figure 2. ◀

Note that each cluster covers only a subset of points, however, the groups of the cluster always include all the nodes of $V$. Given a cluster $C \subseteq V$ and a constant $\nu > 0$, let $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ be the groups obtained by applying Lemma 5 on $C$ with the given value of $\nu$. We let $H_\nu(C) = \cup_{i=1}^{\sigma} \delta(g_i)$ denote the **$\nu$-proper hubs** of the cluster. Notice that the size of $|H_\nu(C)|$ is constant, depending on $\nu$.

Given a cluster $C \subseteq V$ and a constant $\nu > 0$, consider the $\nu$-proper hubs of the cluster, $H_\nu(C)$. We refer to $\text{cost}_{H_\nu}(C) = \sum_{i=1}^{\sigma} \sum_{j=i+1}^{\sigma} \sum_{u \in g_i \cap C, v \in g_j \cap C} d_{H_\nu(C)}(u, v)$ as the **$\nu$-approximate cost** of the cluster. This represents the sum of hub-distances between all pairs of points of $C$ belonging to different groups. The following lemma shows that $\text{cost}_{H_\nu}(C)$ is "almost" equal to $D(C)$, when the value of $\nu$ is sufficiently small.

▶ **Lemma 6.** *For each such cluster $C$ and any $\nu > 0$, $cost_{H_\nu}(C) \leq D(C) \leq (1 + O(\nu))cost_{H_\nu}(C)$.*

The proof is omitted due to page limitations.

To make the presentation of our dynamic programming algorithm simpler, we formulate a problem with the same input and objective as the min-sum $k$-clustering problem, but the cost of clusters is evaluated by $\text{cost}_{H\nu}(C)$ instead of $D(C)$: Given a constant $\nu > 0$ and an edge-weighted tree $T = (V, E)$. In the **Min-Hub $k$-Clustering** problem (**$k$-MHC**), we are asked to partition points $V$ into $k$ clusters $C_1, ..., C_k$ to minimize $\sum_{i=1}^{k} \text{cost}_{H_\nu}(C_i)$.

▶ **Theorem 7.** *Let $\epsilon > 0$. A $(1 + \epsilon)$-approximation for **$k$-MHC** will imply a $(1 + O(\epsilon))$-approximation for $k$-MSC on tree metrics.*

The proof is omitted due to page limitations.

## 2.1 QPTAS for $k$-MHC on Trees with Logarithmic Heights

Theorem 7 tells us that if we try to find a clustering which optimizes the objective of **$k$-MHC**, then the same clustering has a good value for the objective of **$k$-MSC**. Suppose we are given a tree $T = (V, E)$ that has a logarithmic height and a constant $\nu > 0$. Let $OPT$ be the minimum cost of partitioning $V$ into $k$ clusters $C_1, C_2, \cdots, C_k$ with the total cost being $\sum_{i=1}^{k} \text{cost}_{H_\nu}(C_i)$. Given $\epsilon > 0$, we will present a dynamic program that finds a $(1 + \epsilon)$-approximation of $OPT$. This, as a result of Theorem 7, leads to a $(1 + O(\epsilon))$ approximation solution for **$k$-MSC** on trees with logarithmic heights. Then, in the next section, we will extend the dynamic program to cover metrics with bounded treewidth, thereby covering general trees as well.

**Preprocessing.** We assume each node of the tree has a *token* on it and our goal is to cluster the tokens. We may modify the tree by adding dummy edges (with zero weight) and dummy nodes (that do not have tokens). Throughout this section, we refer to a node with a token as a *point* and a node without a token as a *vertex*. By introducing zero-weight edges and nodes, we convert the tree into an equivalent binary tree in which the points are only located on distinct leaves. We repeatedly remove leaves with no tokens until there is no such leaf in the tree. We also repeatedly remove internal vertices (with no token) of degree two by consolidating their incident edges into one edge of the total weight.

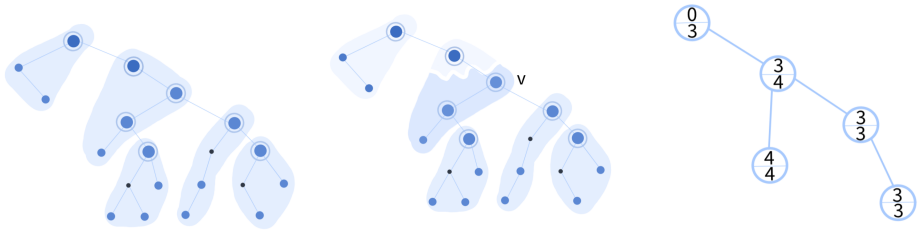



**Figure 3** (a) A cluster and its corresponding groups. (b) The partial cluster with respect to $T_v$. (c) The corresponding backbone tree whose nodes are labelled according to their sizes/weights.

**Cluster, Backbone Tree, and Partial Cluster Types.** Let $\nu > 0$ and consider a cluster $C \subseteq V$. Suppose $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ are the groups by Lemma 5. We define a tree called the **backbone tree** of $C$, with nodes corresponding to groups $g_1, \ldots, g_\sigma$. This tree has edges between nodes whose corresponding groups are connected by an edge. We use $g_i$ to refer to both the group and the corresponding node in the backbone tree. According to Cayley's formula [1], the number of different trees that can be formed by $\tilde{n}$ labeled nodes is $\tilde{n}^{\tilde{n}-2}$. Hence the cluster's backbone tree has one of the types $1, 2, \cdots, \sigma^{\sigma-2}$.

Each cluster $C$ is associated with a pair $(t_b, \vec{w})$ (referred to as the **cluster type** of $C$), where $t_b$ is an integer between 1 and $\sigma^{\sigma-2}$ and represents the type of the cluster's backbone tree, and $\vec{w}$ is a vector representing the weights of each node in the backbone tree, with $\vec{w}[i] = |g_i \cap C|$ being the number of points in the $i$-th group of the cluster; see Figure 3.

The maximum number of ways to assign weights to nodes of a backbone tree is $n^\sigma$, where $n = |V|$. To keep the number of different cluster types manageable, we store the group weights approximately by rounding them to the nearest *threshold value*. This reduces the number of possible ways to assign weights to nodes of a backbone tree to a poly-logarithmic number and so allows for a more compact representation of the cluster types.

▶ **Definition 8.** *Given $\epsilon > 0$, let $\epsilon'$ be $\frac{\epsilon}{c \log n}$. Let **logarithmic threshold values** be $\Phi_{\epsilon,n} = \{\phi_1, \cdots, \phi_\tau\}$ where $\phi_i = i$ for $1 \leq i \leq \lceil \frac{1}{\epsilon'} \rceil$, and for $i > \frac{1}{\epsilon'}$ we have $\phi_i = \lceil \phi_{i-1}(1+\epsilon') \rceil$, and $\phi_\tau = n$. So $\tau = O(\frac{\log n}{\epsilon})$. We define a **mapping** $\phi$ which associates with each value $1 \leq i \leq n$ the minimum threshold value $\phi_j$ for which $i \leq \phi_j$ holds.*

By rounding the weights of groups to the nearest threshold value, the number of different cluster types is reduced to $O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^\sigma))$, where $\sigma = O(1/\nu)$. We will show that, by choosing the number of thresholds appropriately large, the DP solution will have a multiplicative error of at most $1 + O(\epsilon)$ (provided that the tree has a logarithmic height).

For every cluster $C \subset V$ and every node $v \in V$, the part of cluster that falls into $T_v$ is referred to as the *partial cluster of $C$ with respect to $v$*. To represent such a partial cluster, we associate it with a triple $(t_c, \gamma_v, \vec{s}_v)$, where $t_c$ is an integer between 1 and $O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^\sigma))$ and represents the type of the cluster, $\gamma_v$ is the *split group* of the partial cluster and specifies the group that includes the node $v$, and $\vec{s}_v$ is a vector representing the sizes of each group of the partial cluster that intersects with the tree $T_v$, with $\vec{s}_v[i] = |(g_i \cap C) \cap V(T_v)|$ being the number of points in the $i$-th group that intersect with $V(T_v)$; see Figure 3. Similar to the group weights, the group sizes are stored approximately by rounding them to the nearest threshold value. This results in a reduction of the number of partial cluster types to $O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^\sigma))$. Observe that a partial cluster $C$ with respect to root $r$ is actually the full cluster $C$. This means that for every group $i$ in the cluster, the value $\vec{s}_r[i]$ is equal to $\vec{w}[i]$.

We let $\Gamma_v \subseteq \mathbb{C}_\nu$ indicate the groups, called the *inner groups*, of the partial cluster whose nodes are completely contained within the tree node $T_v$. For a specific partial cluster type $\ell$ at $v$, we use the notation $\gamma_v^\ell, \Gamma_v^\ell, \vec{s}_v^\ell$, and $\vec{w}^\ell$ to refer to its split group, inner groups, size, and weight vectors, respectively. It is important to note that both the weight vector $\vec{w}^\ell$ and the inner groups $\Gamma_v^\ell$ can be obtained from the triple $(t_c, \gamma_v, \vec{s}_v)$ that defines $\ell$.

A partial cluster type $\ell$ with respect to a node $v$ is considered **valid** if the following conditions are met: (i) the values of $\vec{s}_v^\ell[i]$ for each group $i$ of $\ell$ are between 0 and $\vec{w}^\ell[i]$, (ii) the value of $\vec{w}^\ell[i]$ for each group $i$ of $\ell$ is less than or equal to $\max\{\nu.\sum_{i'} \vec{w}^\ell[i'], 1\}$ (see Lemma 5), (iii) if $v$ is a leaf node of $T$, then $\gamma_v^\ell$ is a leaf node of the backbone tree of $\ell$ (from the definition of the backbone tree). A partial cluster type $\ell$ is considered a **leaf partial cluster type** at a node $v$ if $\gamma_v^\ell$ is a leaf node of the backbone tree of $\ell$ and $\vec{s}_v^\ell[\gamma_v^\ell] = 1$.

**Edge Load, Partial Cluster Cost, and Cluster Cost.** Consider a cluster $C$ together with its groups $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ and hubs $H_\nu(C)$ , and let $\ell$ be the type of this cluster with respect to $v$. Recall that, vectors $\vec{w}^\ell$ and $\vec{s}_v^\ell$ are used to show the weight and the size (with respect to the tree $T_v$) of the groups within the cluster $C$, and $\gamma_v$ is used to specify the group of the cluster that includes the node $v$. Here, we explain how to compute the $\nu$-approximate cost of the cluster, $\text{cost}_{H_\nu}(C)$, by utilizing the information provided by these vectors.

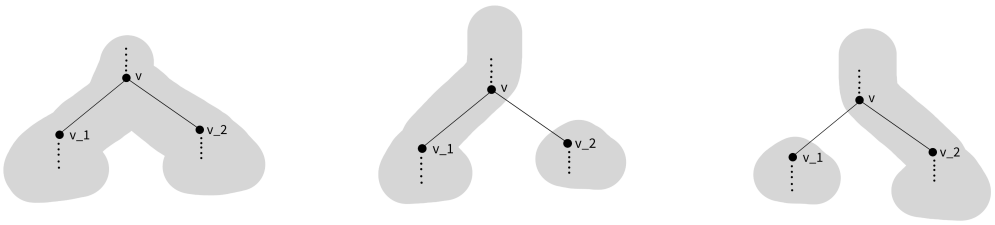
We define the load of edge $e$ with respect to the cluster $C$, its groups $\mathbb{C}_\nu$, and hubs $H_\nu(C)$ to be the number of paths $p_H(u,v)$ that include edge $e$ over all $(u,v) \in X$, where $X = \cup_{i=1}^{\hat{\sigma}} X_i$ and $X_i = \{(u,v) : u \in \hat{g}_i \;\; v \in C \setminus \hat{g}_i\}$. Let $e_v$ denote the edge connecting $v$ to its parent in $T$. The load of edge $e_v$ with respect to $\ell$ can be calculated using the following formula, represented as $load^\ell(e_v)$:

$$load^\ell(e_v) = \underbrace{\Big(\sum_{i=1, i\neq\gamma_v}^{\sigma} \vec{s}_v^\ell[i]\Big) \times \Big(\sum_{i=1}^{\sigma} (\vec{w}^\ell[i] - \vec{s}_v^\ell[i])\Big)}_{\#\text{paths crossing } e_v \text{ s.t. one of its ends is below } \gamma_v} + \underbrace{\vec{s}_v^\ell[\gamma_v] \times \Big(\sum_{i\notin\Gamma_v}^{\sigma} \vec{w}^\ell[i]\Big)}_{\#\text{paths crossing } e_v \text{ s.t. one of its ends is in } \gamma_v}$$

We define and compute the cost of a partial cluster type $\ell$ with respect to a node $v$ (we denote it by $cost_v^\ell$) recursively as follows. For the base case, $cost_v^\ell = 0$, if $v$ is a leaf node. For the recurrence, $cost_v^\ell = cost_{v_1}^\ell + cost_{v_2}^\ell + load^\ell(e_{v_1})w(e_{v_1}) + load^\ell(e_{v_2})w(e_{v_2})$, where $v_1, v_2$ are children of $v$. Note that the union of groups of each cluster always includes the root node $r$ (see Algorithm 1). One can verify that $cost_r^\ell = \text{cost}_{H_\nu}(C)$, if $\ell$ stores the exact weights and sizes of the groups of the cluster. However, here, $\ell$ stores weights and sizes approximately and therefore the edge load $load^\ell(e_v)$ might be overestimated by a factor of $(1+\epsilon')$ (by choosing the number of thresholds appropriately large). In the next section, we will see how this affects our approximation solution and results in a multiplicative error of at most $1 + O(\epsilon)$ (provided that the tree has a logarithmic height).

## Dynamic Program

The Dynamic Program (DP) starts at the leaves of $T$ and works its way up, exploring all possible ways to form clusters. For each node $v$ and each possible configuration $\mathbb{P}_v$ of partial clusters with respect to $v$, there is an entry in the DP table. A configuration $\mathbb{P}_v \in [k]^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^\sigma)\}}$ at node $v$ lists the number of each type of partial cluster covering points within subtree $T_v$. We let $A[v, \mathbb{P}_v]$ store the minimum cost to form a set of partial clusters, which match the configuration $\mathbb{P}_v$, and cover all points in $T_v$. Observe that the number of such subproblems is at most $n^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^\sigma)}$.




(a) (b) (c)

**Figure 4** Consider a node $v$ and its children $v_1$, $v_2$. There are three possible scenarios in which $v$, $v_1$, and $v_2$ may belong to one or two groups of a cluster. (a) is depicting the case where all three nodes are in the same group, (b) is depicting the case that $v$ and $v_1$ are in the same group, (c) is depicting the case that $v$ and $v_2$ are in the same group. Note that the case where all three nodes belong to different groups does not happen due to Algorithm 1.

Consider a node $v$ in the tree $T$. Assume for now that we have access to a table $\lambda[\mathbb{P}_v, \mathbb{P}_{v_1}, \mathbb{P}_{v_2}]$, where $\mathbb{P}_v$ is the configuration at node $v$, and $\mathbb{P}_{v_1}$ and $\mathbb{P}_{v_2}$ are the configurations at its children nodes $v_1$ and $v_2$, respectively. The table $\lambda$ indicates whether the configurations $\mathbb{P}_v$, $\mathbb{P}_{v_1}$, and $\mathbb{P}_{v_2}$ are *consistent*, meaning that there is a solution where the descriptions of partial clusters below nodes $v$, $v_1$, and $v_2$ match the configurations $\mathbb{P}_v$, $\mathbb{P}_{v_1}$, and $\mathbb{P}_{v_2}$, respectively. We shall describe how to compute $\lambda$. We will compute the subproblems $A[v, \mathbb{P}_v]$ in a bottom-up manner: We will compute $A[v, \mathbb{P}_v]$ after we have computed the subproblems $A[v_1, \mathbb{P}_{v_1}]$ and $A[v_2, \mathbb{P}_{v_2}]$ for the children of $v$. The subproblems are computed as follows:

**Base Case.** For every leaf node $v$ and every configuration $\mathbb{P}_v$, set: $A[v, \mathbb{P}_v] = 0$ if there exists a type $\ell$ such that $\mathbb{P}_v[\ell] = 1$ and $\ell$ is a leaf partial cluster at $v$. Otherwise, set $A[v, \mathbb{P}_v] = \infty$.

**Recurrence.** Let $load(v) = \sum_\ell \mathbb{P}_v[\ell] load^\ell(e_v)$. For each internal node $v$ and its children, $v_1, v_2$ and every combination of configurations of $\mathbb{P}_v$ on $v$ and $\mathbb{P}_{v_1}, \mathbb{P}_{v_2}$:

$$A[v, \mathbb{P}_v] = \min_{\mathbb{P}_v, \mathbb{P}_{v_1}, \mathbb{P}_{v_2} : \lambda[\mathbb{P}_v, \mathbb{P}_{v_1}, \mathbb{P}_{v_2}] = True} \sum_{i=1,2} \big( A[v_i, \mathbb{P}_{v_i}] + load(v_i) w(v v_i) \big)$$

The final solution is obtained by finding the minimum value of $A[r, \mathbb{P}_r]$ over all configurations $\mathbb{P}_r$ such that the sum of all $\mathbb{P}_r[\ell]$ values equals $k$; and $\vec{s}_r^{\,\ell}[i] = \vec{w}^{\ell}[i]$ holds, for each partial cluster type $\ell$ with $\mathbb{P}_r[\ell] > 0$, and for all $i$.

**Consistency Constraints.** Consider a node $v$ and its children $v_1, v_2$. Let $P_v = (t_c, \gamma_v, \vec{s}_v), P_{v_1} = (t_{c_1}, \gamma_{v_1}, \vec{s}_{v_1}), P_{v_2} = (t_{c_2}, \gamma_{v_2}, \vec{s}_{v_2})$ be some valid partial cluster types at $v, v_1, v_2$, respectively. We say $P_v$ is consistent with $P_{v_1}$ and $P_{v_2}$ if the following conditions are met:

- **Type Consistency**. The types of $P_v$, $P_{v_1}$, and $P_{v_2}$ must be the same, i.e. $t_c = t_{c_1} = t_{c_2}$.
- **Group Consistency**. The groups of $P_{v_1}$ and $P_{v_2}$ are consistent with those of $P_v$: Recall that $\gamma_v$ indicates the split group of a partial cluster $P_v$ and $\Gamma_v$ indicates the inner groups of $P_v$. Let $\delta_v^{in}$ be the inner groups adjacent to $\gamma_v$ in the backbone tree; $\delta_v^{in} = \delta(\{\gamma_v\}) \cap \Gamma_v$, where $\delta(\{\gamma_v\})$ indicates groups adjacent to $\gamma_v$ (in the backbone tree). Depending on the values of $\gamma_v, \gamma_{v_1}, \gamma_{v_2}$, one of the following cases holds:
  - If $\gamma_v = \gamma_{v_1} = \gamma_{v_2}$ (Figure 4a), then $\delta_{v_1}^{in} \cup \delta_{v_2}^{in} = \delta_v^{in}, \quad \delta_{v_1}^{in} \cap \delta_{v_2}^{in} = \emptyset$.
  - If $\gamma_v = \gamma_{v_1}$ and $\gamma_{v_2} \in \delta_v^{in}$ (Figure 4b), then $\delta_{v_1}^{in} = \delta_v^{in} \setminus \{\gamma_{v_2}\}, \quad \delta_{v_2}^{in} = \delta(\{\gamma_{v_2}\}) \setminus \{\gamma_v\}$.
  - If $\gamma_v = \gamma_{v_2}$ and $\gamma_{v_1} \in \delta_v^{in}$ (Figure 4c), then $\delta_{v_2}^{in} = \delta_v^{in} \setminus \{\gamma_{v_1}\}, \quad \delta_{v_1}^{in} = \delta(\{\gamma_{v_1}\}) \setminus \{\gamma_v\}$.

- **Size Consistency**. The group sizes of $P_1$ and $P_2$ are consistent with those of $P$. Depending on the values of $\gamma_v, \gamma_{v_1}, \gamma_{v_2}$, one of the following cases holds:
  - If $\gamma_v = \gamma_{v_1} = \gamma_{v_2}$, then we ensure that $\phi\big(\vec{s}_{v_1}[\gamma_{v_1}] + \vec{s}_{v_2}[\gamma_{v_2}]\big) = \vec{s}_v[\gamma_v]$.
  - If $\gamma_v = \gamma_{v_1}$ and $\gamma_{v_2} \in \delta_v^{in}$, then we ensure that $\vec{s}_{v_2}[\gamma_{v_2}] = w[\gamma_{v_2}]$ and $\vec{s}_{v_1}[\gamma_{v_1}] = \vec{s}_v[\gamma_v]$.
  - If $\gamma_v = \gamma_{v_2}$ and $\gamma_{v_1} \in \delta_v^{in}$, then we ensure that $\vec{s}_{v_1}[\gamma_{v_1}] = w[\gamma_{v_1}]$ and $\vec{s}_{v_2}[\gamma_{v_2}] = \vec{s}_v[\gamma_v]$.

Note that the case that $\gamma_{v_1} = \gamma_{v_2}, \gamma_v \neq \gamma_{v_1}$ is impossible since each group of the cluster covers a connected subtree. Furthermore, the case when $\gamma_{v_1} \in \delta_v^{in}$ & $\gamma_{v_2} \in \delta_v^{in}$ is impossible using the fact that there is no point on the internal node $v$ (see the preprocessing step).

The value of $\lambda[\mathbb{P}_v, \mathbb{P}_{v_1}, \mathbb{P}_{v_2}]$ is calculated recursively for every combination of configurations of $v$ and its children, $v_1, v_2$. For the base case $\lambda[\vec{0}, \vec{0}, \vec{0}] =$ True. Let $\mathbb{P}_v - P_v$ indicate the configuration of $\mathbb{P}_v$ with one less partial cluster of type $P_v$. For the recurrence, we consider all possible *consistent* valid partial cluster types $P_v$, $P_{v_1}$ and $P_{v_2}$

$$\lambda[\mathbb{P}_v, \mathbb{P}_{v_1}, \mathbb{P}_{v_2}] = \bigvee_{\forall \text{ consistent } P_v, P_{v_1}, P_{v_2}} \lambda[\mathbb{P}_v - P_v, \mathbb{P}_{v_1} - P_{v_1}, \mathbb{P}_{v_2} - P_{v_1}]$$

### Analysis

In our DP, configurations store the rounded sizes (and weights) of the partial clusters' groups. To ensure consistency between the sizes of the groups at node $v$ and its children $v_1$ and $v_2$, we allow the size of the group at $v$ to be a $(1+\epsilon')$ upper bound for the combined size of the groups at $v_1$ and $v_2$. This results in a multiplicative error of at most $(1+\epsilon')$ in the calculation of the edges' loads and so the cost of the partial clusters at each node of the tree when the sizes (weights) of merged partial clusters are rounded. Given that the height of the tree is $h$, it is not difficult to see that our dynamic programming approach finds a solution that is an $(1+\epsilon')^h$-approximation to the problem.

The number of possible configurations $\mathbb{P}_v$ for each node $v$ is at most $n^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^{\sigma}))}$, resulting in $n^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^{\sigma}))}$ dynamic program table entries. To compute each entry in the DP table, we iterate over all consistent configurations at $v$, $v_1$, and $v_2$, which takes $n^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^{\sigma}))}$ time. Hence, the overall running time of the algorithm is $n^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon'})^{\sigma}))}$, which is still a quasi-polynomial time complexity in $n$. By setting $\epsilon' = \frac{\epsilon}{\log n}$ in the threshold mapping, the algorithm finds a $(1+\epsilon)$ approximation solution in time $n^{O(\sigma^{\sigma-2}(\frac{\log n}{\epsilon})^{\sigma+1}))}$.

▶ **Theorem 9.** *There is a QPTAS for the* ***k-MSC*** *problem on trees with logarithmic heights.*

## 3 The $k$-MSC Problem in Metrics of Bounded Treewidth

In this section, we extend our algorithm from Section 2 to metrics of bounded treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (V', E')$ on a new set of nodes $V'$, where each $i \in V'$ corresponds to a subset $b_i$, called a *bag*, of vertices of $V$ with the following properties: (i) $\cup_{i \in V'} b_i = V$, (ii) for every edge $uv \in E$, there exists a bag $t$ of $T$ such that $b_t$ contains both $u$ and $v$, (iii) if $b_i, b_j$ contain vertex $v$ then every bag on the path between $i$ and $j$ in $T$ contains $v$. The *width* of a tree decomposition $T$ is the size of the largest bag of $T$ minus one; this is $\max_{i \in V'}(|b_i| - 1)$. The *treewidth* of a graph $G$ is the minimum width over all possible tree decompositions of G. The authors of [5] showed that any graph $G = (V, E)$ with treewidth $f$ has a tree decomposition $T$ of width at most $3f + 2$ that has the following two extra properties: (i) $T$ is binary, (ii) the height of $T$ is $O(\log |V|)$.

Given a graph $G = (V, E)$ with a treewidth of $f'$, we create a binary decomposition tree $T = (V', E')$ with a width of no more than $3f' + 2$ and a height of logarithmic in $|V|$ (see [5]). Let $f$ be the width of $T$. We will refer to $G$ as the graph and $T$ as the tree. We will refer to vertices in $V$ as *nodes* and vertices in $V'$ as *bags*. We will refer to edges in $G$ as *edges* and

edges in $T$ as *super-edges*. Let $T$ be rooted at an arbitrary bag $r \in V'$. We use $T_b$ to denote the *subtree* rooted at the bag $b$, $V'(T_b)$ to denote the bag set of $T_b$, and $E'(T_b)$ to denote the super-edge set of $T_b$. Each node $u \in V$ can appear in multiple bags of $V'$, and these bags form a subtree of $T$. To ensure that each point is covered only once, we consider the point as a *token* placed at the node. We place the token of a node at the bag closest to the root of $T$ that contains the node. This bag is marked as the one containing the point.

We further modify the tree to make sure that (i) only the leaf bags contain the tokens and (ii) each bag contains at most one token: for any bag A that violates these two rules, create two new bags B and C that are identical copies of A. Move one of the tokens from the original bag A to bag C and place any remaining tokens in bag B. Connect the children of the original bag A to the newly created bag B. Connect both bags B and C to A. Finally, we remove all leaf bags without any tokens. This process results in a binary tree decomposition with a height of $O(\log n)$. We call this tree decomposition with these properties the **proper tree decomposition** of the graph. For each point $u \in V$, we let $B_u \in V'$ denote the bag that contains point $u$. For each $C \subseteq V$, let $B_C = \{B_u : u \in C\}$.

Consider a mapping $p : V' \to V'$ that maps each bag to its parent bag and maps $r$ to itself. Let $e_b$ be the super-edge between $b$ and $p(b)$ in $T$. The edges $(s,t)$ where $s \in b$ and $t \in p(b)$ are referred to as the **bridge-edges** with respect to the super-edge $e_b$. We use the notation $e_b^{s,t}$ to refer to these edges. An edge between such vertices $s \in b$ and $t \in p(b)$ is added in $G$ with a weight of $d(s,t)$ if it does not already exist. For any pair of points $u$ and $v$ in $V$, one can verify that there exists a path between $u \in B_u$ and $v \in B_v$ in the tree $T$ consisting only of bridge-edges over the super-edges which is equivalent to the shortest path between $u$ and $v$ in the graph. This path connects the bags $B_u$ and $B_v$ in $T$ and only uses the bridge-edges over the super-edges of the unique path connecting these bags in the tree. The length of this path is equal to $d(u,v)$, the distance between $u$ and $v$ in the graph $G$. This path is referred to as $p_B(u,v)$.

For each bag $b \in V'$, let $V'_b = \cup_{i \in V'(T_b)} b_i$ denote the union of nodes in bags of $V'(T_b)$. For a tree decomposition $T = (V', E')$ and a subset of bags $\hat{V} \subseteq V'$, we use $\delta(\hat{V}) = \{b_i \in \hat{V} : b_i b_j \in E' \ \&\ b_j \notin \hat{V}\}$ to denote the *border bags* of $\hat{V}$. The proof of the following lemma is analogous to that of Lemma 5.

▶ **Lemma 10.** *Given a graph $G = (V, E)$ of bounded treewidth, a proper tree decomposition $T = (V', E')$ of $G$, a set of points $C \subseteq V$, for any $\nu > 0$, there exists a partition of $V'$ into a set of groups $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ such that all of the following properties hold: (i) The subgraph induced by each group $g \in \mathbb{C}_\nu$ is connected in $T$. (ii) For each group $g \in \mathbb{C}_\nu$, $|g \cap B_C| \in [1, \max\{1, \nu|C|\}]$. (iii) $\sigma = O(1/\nu)$. (iv) $\forall g \in \mathbb{C}_\nu, \quad |\delta(g)| = O(1/\nu)$.*

Let $\nu > 0$. Consider a cluster $C \subseteq V$. Let $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ be the groups obtained by Lemma 10. For each such cluster $C$ and any constant $\nu > 0$, we let $H_\nu(C) = \cup_{i=1}^{\sigma} \cup_{j \in \delta(g_i)} b_j$ denote the *border hubs* of the cluster and $\text{cost}_{H_\nu}(C) = \sum_{i=1}^{\sigma} \sum_{j=i+1}^{\sigma} \sum_{u \in V(g_i) \cap C, v \in V(g_j) \cap C} d_{H_\nu(C)}(u,v)$ be the *$\nu$-approximate cost* of the cluster. Notice that for any two points $u$ and $v$ in $C$ that belong to different groups of $\mathbb{C}_\nu$, the path $p_B(u,v)$ passes through the hubs $H_\nu(C)$, implying $d(u,v) = d_{H_\nu(C)}(u,v)$. The proof of the following is analogous to that of Theorem 7.

▶ **Theorem 11.** *Given $\epsilon > 0$, a $(1+\epsilon)$-approximation for **k-MHC**, will imply a $(1 + O(\epsilon))$-approximation for **k-MSC** on bounded treewidth graphs.*

## 3.1 QPTAS for $k$-MHC on Graphs of Bounded Treewidth

Given $\nu > 0$ and a graph $G(V, E)$ that has a *proper decomposition tree* $T = (V', E')$ with a logarithmic height and a treewidth of $f$. Let $OPT$ be the minimum cost of partitioning $V$ into $k$ clusters $C_1, C_2, \cdots, C_k$ with the total cost being $\sum_{i=1}^{k} \text{cost}_{H_\nu}(C_i)$. Given $\epsilon > 0$, we will present a dynamic program that finds a $(1+\epsilon)$ approximation of $OPT$. This, as a result of Theorem 11, leads to a $(1 + O(\epsilon))$ approximation solution for the ***k*-MSC** problem.

Consider a cluster $C \subseteq V$. Let $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ be the groups obtained by Lemma 10 on $C$. We define a *backbone tree* associated with the cluster $C$. This tree is made up of $O(1/\nu)$ nodes that correspond to the groups of $\mathbb{C}_\nu$ and there are edges between the nodes in the tree if the corresponding groups in $\mathbb{C}_\nu$ are connected by a super-edge in the tree $T$. A *cluster type* is defined as a node-weighted backbone tree where each node in the tree is assigned a weight from the threshold values $\Phi_{\epsilon,n}$ (see Definition 8) which represents the number of points in the corresponding group rounded up to the nearest threshold value.

For each cluster $C$ and bag $b$ in tree $T$, we associate a partial cluster type to it. This is represented by a triple $(t_c, \gamma_b, \vec{s}_b)$ and includes: the type of the cluster, $t_c$; the group of the cluster that has bag $b$, $\gamma_b$; and a vector $\vec{s}_b$, where $\vec{s}_b[i]$ denotes the number of points in the $i$th group located in tree $T_b$. It is not hard to verify that the number of possible partial clusters is $O(\sigma^{\sigma-2} \log^{\sigma}_{(1+\epsilon')} n) = O((\frac{\log n}{\epsilon})^{\sigma+1})$, where we fix $\sigma = O(1/\nu)$.

We use $\ell \in \{1, 2, \ldots, O((\frac{\log n}{\epsilon})^{\sigma+1})\}$ to refer to a specific partial cluster type. A partial cluster type $\ell$ with respect to a vertex $b$ is considered **valid** if: the values of $\vec{s}^{\ell}_v[i]$ for each group $i$ of $\ell$ are between 0 and $\vec{w}^{\ell}[i]$, the value of $\vec{w}^{\ell}[i]$ for each group $i$ of $\ell$ is less than or equal to $\max\{\nu. \sum_{i'} \vec{w}^{\ell}[i'], 1\}$, and if $v$ is a leaf vertex of $T$, then $\gamma^{\ell}_v$ is a leaf node of the backbone tree of $\ell$. A partial cluster type $\ell$ is considered a **leaf partial cluster type** at a vertex $b$ if $\gamma^{\ell}_b$ is a leaf node of the backbone tree of $\ell$ and $\vec{s}^{\ell}_b[\gamma^{\ell}_v] = 1$.

Consider a cluster $C$ together with its groups $\mathbb{C}_\nu = \{g_1, \ldots, g_\sigma\}$ and hubs $H_\nu(C)$, and let $\ell$ be the type of this cluster with respect to bag $b$. Here, we explain how to compute the $\nu$-approximate cost of the cluster, $\text{cost}_{H_\nu}(C)$. Let $X = \cup_{i=1}^{\sigma} X_i$ and $X_i = \{(u, v) : u \in V(g_i) \;\; v \in C \setminus V(g_i)\}$. Let $e_b$ denote the super edge connecting $b$ to its parent bag $p(b)$ in $T$. We define load of a bridge-edge $e^{s,t}_b$ with respect to the cluster $C$, its groups $\mathbb{C}_\nu$, and hubs $H_\nu(C)$ to be the number of paths $p_B(u, v)$ that contain this edge over all $\{u, v\} \in X$. We use $load^{\ell}(e^{s,t}_b)$ to represent the load of bridge-edge $e^{s,t}_b$ with respect to partial cluster type $\ell$ and bag $b$. Similarly, we use $load^{\ell}(e_b)$ to represent the load of super-edge $e_b$ with respect to partial cluster type $\ell$ and bag $b$.

Similarly to the case of the tree, the load of the super-edge $e_b$ with respect to $\ell$ can be calculated using the following formula: $load^{\ell}(e_b) = \big(\sum_{i=1, i\neq\gamma_b}^{\sigma} \vec{s}^{\ell}_b[i]\big) \times \big(\sum_{i=1}^{\sigma} (\vec{w}^{\ell}[i] - \vec{s}^{\ell}_b[i])\big) + \vec{s}^{\ell}_b[\gamma_b] \times \big(\sum_{i\notin\Gamma_b}^{\sigma} \vec{w}^{\ell}[i]\big)$. Note that $load^{\ell}(e_b)$ computes the number of paths $p_{H_\nu(C)}(u, v)$ in $G$ that cross the cut-set $(b, p(b))$ for all pairs of points $(u, v)$ in the set $X$.

When computing the cost of a cluster type, it is necessary to take into account the load among the bridge-edges. However, the load of a bridge-edge cannot be calculated simply from the sizes and weights of the groups within the cluster, unlike the load of the super-edges.

To address this issue, for each partial cluster type $\ell$ and each $b$, we have defined a vector $\psi^{\ell}_b$ with a dimension of $f^2$ (where $f$ is the treewidth of the graph), that $\psi^{\ell}_b[e^{s,t}_b]$ specifies the load of each bridge-edge $e^{s,t}_b$ with respect to $\ell$. One can now compute the cost of a partial cluster $\ell$ at bag $b$, denoted by $cost^{\ell}_b$, recursively as follows. For the base case, $cost^{\ell}_b = 0$, if $b$ is a leaf bag. For the recurrence, $cost^{\ell}_b = cost^{\ell}_{b_1} + cost^{\ell}_{b_2} + \sum_{\{s,t\}\in b_1\times b} \psi^{\ell}_{b_1}(e^{b_1}_{s,t}) w(e^{b_1}_{s,t}) + \sum_{\{s,t\}\in b_2\times b} \psi^{\ell}_{b_2}(e^{b_2}_{s,t}) w(e^{b_2}_{s,t})$, where $b_1, b_2$ are children of $b$.

We could attach $\psi_b^\ell$ (with a dimension of $f^2$ which approximately stores the flow of the bridge edges) to the vectores we store for each cluster type $\ell$ to obtain a QPTAS for the problem on graphs with bounded treewidth. However, this QPTAS cannot be extended to include graphs with bounded highway dimension or graphs with bounded doubling dimensions (as $f$ becomes logarithmic in these cases). To address this issue, in the next section we propose that at each bag $v$, it is sufficient to store information about the total flow of the partial clusters that passes through the bridge edges, in addition to the information about the type of partial cluster covering the points within the subtree. This eliminates the need to separately store the flow of each partial cluster.

### Dynamic program

The Dynamic Program (DP) traverses $T$ starting at the leaves and moving upward and considers all ways partial clusters can be made. At each bag $b$, a configuration $< b, \mathbb{P}_b, \psi_b >$ is defined. In this configuration, $\mathbb{P}_b$ specifies the number of partial clusters of each type covering points within $T_b$, and $\psi_b$ specifies the total load for each bridge-edge over all the partial cluster types $\ell$ specified in $\mathbb{P}_b$; namely, $\psi_b = \sum_\ell \mathbb{P}_b[\ell].\psi_b^\ell$.

**Valid Configuration.** The validity check of a configuration involves ensuring the feasibility of the load distributions among partial clusters. For a given bag $b$ and configuration $(\mathbb{P}_b, \psi_b)$, we can use the loads of super edges to get the total loads crossing $b$: $\Psi_b = \sum_\ell \mathbb{P}_b[\ell] load^\ell(e_b)$. We say the configuration $(\mathbb{P}_b, \psi_b)$ is *valid* if the following holds: $\phi(\Psi_b) = \phi\left(\sum_{e^b_{s,t} \in b \times p(b)} \psi[e^b_{s,t}]\right)$; this is, the total load of the partial clusters crossing super-edge $e_b$ (this can be obtained via $\mathbb{P}_b$ as described in the previous section) must be equal to the total load of the partial clusters crossing all the bridge-edges with respect to the super-edge $e_b$. Note that when $b$ is a leaf, this condition implies that, $\phi(\sum_{e^b_{s,t} \in b \times p(b)} \psi[e^b_{s,t}]) = \phi(\sum_i w[i] - 1)$.

Assume for now that we have access to an inner table $\varphi[(\mathbb{P}, \psi), (\mathbb{P}_1, \psi_1), (\mathbb{P}_2, \psi_2)]$ that for every combination of configurations of $(\mathbb{P}, \psi)$ on $b$ and $(\mathbb{P}_1, \psi_1), (\mathbb{P}_2, \psi_2)$ on its children, $b_1, b_2$, indicates whether they are *consistent* or not. The representation of $\perp$ is used to indicate the empty configurations for handling the cases when $b$ is a leaf or has one child.

Let $A[b, \mathbb{P}_b, \psi_b]$ be the minimum cost solution for subproblem $< b, \mathbb{P}_b, \psi_b >$ in which points in $V'_b$ are covered by a set of partial clusters whose types (and loads) are consistent with the configuration $\mathbb{P}_b, \psi_b$ (recall that $V'_b = \cup_{i \in T_b} b_i$).

We will compute the subproblems $A[b, \mathbb{P}_b, \psi_b]$ in a bottom-up manner:

**Base Case.** For each leaf vertex $b$: $A[b, \mathbb{P}_b, \psi_b] = 0$ if $\varphi[(\mathbb{P}_b, \psi_b), \perp, \perp] =$ True and otherwise it is $\infty$.

**Recurrence.** For each internal vertex $b$ and its children, $b_1, b_2$:

$$A[b, \mathbb{P}_b, \psi_b] = \min_{\varphi[(\mathbb{P}_b, \psi_b), (\mathbb{P}_{b_1}, \psi_{b_1}), (\mathbb{P}_{b_2}, \psi_{b_2})] = True} \left\{ \sum_{i=1,2} \left( A[b_i, \mathbb{P}_{b_i}, \psi_{b_i}] + \sum_{\{s,t\} \in b_i \times b} \psi_b[e^{b_i}_{s,t}] w(e^{b_i}_{s,t}) \right) \right\}$$

The case of $b$ having one child is similar. The final solution is obtained by finding the minimum value of $A[b, \mathbb{P}_b, \psi_b]$ over all valid configurations $< \mathbb{P}_b, \psi_b >$ such that the sum of all $\mathbb{P}_b[\ell]$ values equals $k$.

### Consistency Constraints

Consider a bag $b$ and its two children $b_1$ and $b_2$. Let $< \mathbb{P}_b, \psi_b >$, $< \mathbb{P}_{b_1}, \psi_{b_1} >, and < \mathbb{P}_{b_2}, \psi_{b_2} >$ be some configurations at $b$, $b_1$, and $b_2$, respectively. To check the consistency of them, there are two steps to follow: (1) verify the *feasibility of partial cluster types*; if the types of the partial clusters in $\mathbb{P}_b$ match those in $\mathbb{P}_{b_1}$ and $\mathbb{P}_{b_2}$. (2) ensure the *feasibility of load distributions*; if the load distribution of the clusters in $\psi_b$ aligns with the load distributions of the clusters in $\psi_{b_1}$ and $\psi_{b_2}$. If these two conditions are met, $\varphi[(\mathbb{P}_b, \psi_b), (\mathbb{P}_{b_1}, \psi_{b_1}), (\mathbb{P}_{b_2}, \psi_{b_2})]$ will be set to True. Otherwise, it will be set to False.

**Feasibility of Partial Cluster Types.** Here we check if there is a solution where the descriptions of partial clusters below nodes $b$, $b_1$, and $b_2$ match the configurations $\mathbb{P}_b$, $\mathbb{P}_{b_1}$, and $\mathbb{P}_{b_2}$, respectively. This step guarantees that the final clustering covers all the points and is therefore a valid solution. This check is very similar to the consistency verification we performed in the case of the tree. There are three cases, depending on whether $b$ is a *leaf*, a bag with *one child*, or a bag with *two children*:

- when $b$ is a leaf: $\mathbb{P}_b[\ell] = 1$ must hold for some $\ell$, where $\ell$ is a leaf partial cluster at b.
- when $b$ has one child, say bag $b_1$: since there is no point (token) on internal bags, $b$ and $b_1$ must belong to the same group. In this case, we must ensure the following: $t_b = t_{b_1}$ (*type consistency*); $\gamma_b = \gamma_{b_1}, \delta_b^{in} = \delta_{b_1}^{in}$ (*group consistency*); and $\vec{s}_b = \vec{s}_{b_1}$ (*size consistency*).
- when $b$ has two children, $b_1, b_2$. Let $P = (t_c, \gamma_b, \vec{s}_b), P_1 = (t_{c_1}, \gamma_{b_1}, \vec{s}_{b_1}), P_2 = (t_{c_2}, \gamma_{v_2}, \vec{s}_{b_2})$ be considered partial cluster types at $b, b_1, b_2$, respectively. Note that the type of a cluster is made up of backbone tree $t_b$ and weights $\vec{w}$. Recall that similar to trees, $\delta(\{\gamma_b\})$ stands for the adjacent bags of $\gamma_b$ and $\delta_b^{in}$ stands for the adjacent bags of $\gamma_b$ inside $T_b$. We say the partial cluster type $P$ (with respect to $T_b$) is consistent with the two partial clusters $P_1$ and $P_2$ (with respect to $T_{b_1}$ and $T_{b_2}$, respectively) if the following holds: (i) (*type consistency*) $t_c = t_{c_1} = t_{c_2}$. (ii)(*group consistency*) If $\gamma_b = \gamma_{b_1} = \gamma_{b_2}$, then we ensure that $\delta_{b_1}^{in} \cup \delta_{b_2}^{in} = \delta_b^{in}$ and $\delta_{b_1}^{in} \cap \delta_{b_2}^{in} = \emptyset$. If $\gamma_b = \gamma_{b_1}$ and $\gamma_{b_2} \in \delta_b^{in}$, then we ensure that $\delta_{b_1}^{in} = \delta_b^{in} \setminus \{\gamma_{b_2}\}$ and $\delta_{b_2}^{in} = \delta(\{\gamma_{b_2}\}) \setminus \{\gamma_b\}$. If $\gamma_b = \gamma_{b_2}$ and $\gamma_{b_1} \in \delta_b^{in}$, then we ensure that $\delta_{b_2}^{in} = \delta_b^{in} \setminus \{\gamma_{b_1}\}$ and $\delta_{b_1}^{in} = \delta(\{\gamma_{b_1}\}) \setminus \{\gamma_b\}$. (iii) (*size consistency*) If $\gamma_b = \gamma_{b_1} = \gamma_{b_2}$, then we ensure that $\phi\big(\vec{s}_{b_1}[\gamma_{b_1}] + \vec{s}_{b_2}[\gamma_{b_2}]\big) = \vec{s}_b[\gamma_b]$. If $\gamma_b = \gamma_{b_1}$ and $\gamma_{b_2} \in \delta_b^{in}$, then we ensure that $\vec{s}_{b_2}[\gamma_{b_2}] = w[\gamma_{b_2}]$ and $\vec{s}_{b_1}[\gamma_{b_1}] = \vec{s}_b[\gamma_b]$. If $\gamma_b = \gamma_{b_2}$ and $\gamma_{b_1} \in \delta_b^{in}$, then we ensure that $\vec{s}_{b_1}[\gamma_{b_1}] = w[\gamma_{b_1}]$ and $\vec{s}_{b_2}[\gamma_{b_2}] = \vec{s}_b[\gamma_b]$.

For every combination of configurations on $b$ and its children, $b_1, b_2$, $\lambda[\mathbb{P}_b, \mathbb{P}_{b_1}, \mathbb{P}_{b_2}]$ is computed recursively as below. For the base case $\lambda[\vec{0}, \vec{0}, \vec{0}] =$ True. For the recurrence, we consider all possible *consistent* partial cluster types $P_b$, $P_{b_1}$ and $P_{b_2}$

$$\lambda[\mathbb{P}_b, \mathbb{P}_{b_1}, \mathbb{P}_{b_2}] = \bigvee_{\forall \text{ consistent } P_b, P_{v_1}, P_{v_2}} \lambda[\mathbb{P}_b - P_b, \mathbb{P}_{b_1} - P_{b_1}, \mathbb{P}_{b_2} - P_{b_1}]$$

where $\mathbb{P}_b - P_b$ indicates the configuration of $\mathbb{P}_b$ with one less partial cluster of type $P_b$.

**Feasibility of Load Distributions.** This ensures that the sum of all flows through the bridge edges into bag $b$ and the sum of all flows out of it are consistent, and that the flow originates only from points that have tokens. This confirms the accuracy of the solution cost calculated using these bridge-edge load distributions. There are three cases, depending on whether $b$ is a leaf, a bag with one child, or a bag with two children:

- when $b$ is a leaf. Suppose $y \in b$ is the only point of bag $b$, we must ensure that: $\forall st : s \in b, t \in p(b), s \neq y, \psi[e_{s,t}^b] = 0$

- when $b$ has one child, say $b_1$. Loads of configurations $\psi_b, \psi_{b_1}$ are consistent if and only if, for each vertex of $b$, the load coming from $b_1$ into each vertex of $b$ is equal to the load going upwards, formulated as following: $\forall t \in b. \quad \sum_{s \in b_1} \psi[e^{b_1}_{s,t}] = \sum_{u \in p(b)} \psi[e^{b}_{t,u}]$
- when $b$ has two children, $b_1, b_2$. For each $t \in b$ let $L_t$ be $\sum_{s \in b_1} \psi[e^{b_1}_{s,t}]$, $R_t$ be $\sum_{s \in b_1} \psi[e^{b_2}_{s,t}]$, $U_t$ be $\sum_{s \in p(b)} \psi[e^{b}_{t,s}]$. Load vectors of configurations $\psi_b, \psi_{b_1}, \psi_{b_2}$ are consistent if and only if for each $u \in b_b$ one of the following constraints must hold: $L_b + R_b = U_b$ or $|L_b - R_b| = U_b$.

**Proof of Theorem 1.** There are $O((\frac{\log n}{\epsilon})^{\sigma+1})$ possible partial clusters, so the number of subproblem configurations, $\mathbb{P}_b$, at bag $b$ is $n^{O((\frac{\log n}{\epsilon})^{\sigma+1})}$. The number of the possible values for $\psi$, is $n^{f^2}$, resulting in a number of DP table entries of $n^{O(f^2+(\frac{\log n}{\epsilon})^{\sigma+1})}$.

Deciding configurations $(\mathbb{P}_b, \psi_b), (\mathbb{P}_{v_b}, \psi_{b_1}), (\mathbb{P}_{b_2}, \psi_{b_2})$ are consistent requires iterating over all consistent configurations which are at most equal $n^{O(f^2+(\frac{\log n}{\epsilon})^{\sigma+1})}$. Therefore the running time is $n^{O(f^2+(\frac{\log n}{\epsilon})^{\sigma+1})}$, which is quasi-polynomial in $n$. Notice that even if treewidth is poly-logarithmic, the running time stays quasi-polynomial.

We lose a factor of $(1+\epsilon/\log n)$ when computing $A[b, \mathbb{P}_b]$ at each level of recursion. Since the height of the tree is at most $c \log n$, the approximation factor of the solution is $1+\epsilon$. ◀

## 4 Bounded Doubling, Highway Dimension, and Minor-Free Metrics

We assume that the aspect ratio of a given metric in a $k$-MSC instance is polynomially bounded (the details are omitted). We use our QPTAS for $k$-MSC on graphs with bounded treewidth as a black box and combine it with embeddings into polylogarithmic-treewidth graphs [7, 10, 14] to develop QPTASs for $k$-MSC on metric spaces with bounded doubling dimension[1], bounded highway dimension, and minor-free metrics. The details are omitted in this version of the paper.

## References


1 Martin Aigner and Günter M. Ziegler. Cayley's formula for the number of trees. *Proofs from THE BOOK*, pages 201–206, 2010. doi:10.1007/978-3-642-00856-6_30.

2 Sandip Banerjee, Rafail Ostrovsky, and Yuval Rabani. Min-Sum Clustering (With Outliers). In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2021)*, volume 207 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.16.

3 Yair Bartal, Moses Charikar, and Danny Raz. Approximating min-sum k-clustering in metric spaces. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 11–20, 2001.

4 Babak Behsaz, Zachary Friggstad, Mohammad R Salavatipour, and Rohit Sivakumar. Approximation algorithms for min-sum k-clustering and balanced k-median. *Algorithmica*, 81:1006–1030, 2019.


[1] To obtain a QPTAS for Euclidean Min-Sum clustering, we could adopt the approach we proposed for tree-like metrics. This involves using a $(1+\epsilon)$-reduction from Euclidean Min-Sum clustering to Euclidean Min-Hub clustering, achieved by placing hubs of constant size in suitable locations for each cluster. We could then apply Arora's scheme to get a QPTAS for Euclidean Min-Hub clustering, with cluster types determined by the backbone structure of the hubs. We skip the details since this can be derived from the reduction from doubling metrics to bounded treewidth metrics.

5 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM Journal on Computing*, 27(6):1725–1746, 1998. doi:10.1137/S0097539795289859.

6 Vincent Cohen-Addad, CS Karthik, and Euiwoong Lee. On approximability of clustering problems without candidate centers. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2635–2648. SIAM, 2021.

7 Vincent Cohen-Addad, Hung Le, Marcin Pilipczuk, and Michał Pilipczuk. Planar and minor-free metrics embed into metrics of polylogarithmic treewidth with expected multiplicative distortion arbitrarily close to 1. *arXiv preprint arXiv:2304.07268*, 2023. URL: https://arxiv.org/abs/2304.07268.

8 Artur Czumaj and Christian Sohler. Small space representations for metric min-sum k-clustering and their applications. In *Proceedings of the 24th Annual Conference on Theoretical Aspects of Computer Science*, STACS'07, pages 536–548, Berlin, Heidelberg, 2007. Springer-Verlag.

9 Wenceslas Fernandez de la Vega, Marek Karpinski, Claire Mathieu, and Yuval Rabani. Approximation schemes for clustering problems. In *STOC '03*, 2003.

10 Andreas Emil Feldmann, Wai Shing Fung, Jochen Könemann, and Ian Post. A $(1+\varepsilon)$-embedding of low highway dimension graphs into bounded treewidth graphs. *SIAM Journal on Computing*, 47(4):1667–1704, 2018. doi:10.1137/16M1067196.

11 Nili Guttmann-Beck and Refael Hassin. Approximation algorithms for min-sum p-clustering. *Discrete Applied Mathematics*, 89(1-3):125–142, 1998.

12 Viggo Kann, Sanjeev Khanna, Jens Lagergren, and Alessandro Panconesi. On the hardness of approximating max k-cut and its dual. *Chicago J Theoret Comput Sci*, May 1997.

13 Sartaj Sahni and Teofilo F. Gonzalez. P-complete approximation problems. *Journal of the ACM (JACM)*, 23:555–565, 1976.

14 Kunal Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 281–290, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007399.

# Algorithms for Computing Maximum Cliques in Hyperbolic Random Graphs

**Eunjin Oh**
Department of Computer Science, POSTECH, Pohang, South Korea

**Seunghyeok Oh**
Department of Computer Science, POSTECH, Pohang, South Korea

## Abstract

In this paper, we study the maximum clique problem on hyperbolic random graphs. A hyperbolic random graph is a mathematical model for analyzing scale-free networks since it effectively explains the power-law degree distribution of scale-free networks. We propose a simple algorithm for finding a maximum clique in hyperbolic random graph. We first analyze the running time of our algorithm theoretically. We can compute a maximum clique on a hyperbolic random graph $G$ in $O(m+n^{4.5(1-\alpha)})$ expected time if a geometric representation is given or in $O(m+n^{6(1-\alpha)})$ expected time if a geometric representation is not given, where $n$ and $m$ denote the numbers of vertices and edges of $G$, respectively, and $\alpha$ denotes a parameter controlling the power-law exponent of the degree distribution of $G$. Also, we implemented and evaluated our algorithm empirically. Our algorithm outperforms the previous algorithm [BFK18] practically and theoretically. Beyond the hyperbolic random graphs, we have experiment on real-world networks. For most of instances, we get large cliques close to the optimum solutions efficiently.

**2012 ACM Subject Classification** Theory of computation → Random network models

**Keywords and phrases** Maximum cliques, hyperbolic random graphs

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.85

**Related Version** *Full Version*: `https://arxiv.org/abs/2306.16775`

**Supplementary Material** *Software (Source Code)*: `https://github.com/Menborong/HRG_maxClique` archived at `swh:1:dir:e721e2b75d1878c69a07cb1a383a6eff65dcfc5b`

**Funding** This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.RS-2023-00209069).

## 1 Introduction

Designing algorithms for analyzing large real-world networks such as social networks, World Wide Web, or biological networks is a fundamental problem in computer science that has attracted considerable attention in the last decades. To deal with this problem from the theoretical point of view, it is required to define a mathematical model for real-world networks. For this purpose, several models have been proposed. Those models are required to replicate the salient features of real-world networks. One of the most salient features of real-world networks is *scale-free*. In general, a graph is considered as a scale-free network if its diameter is small, one connected component has large size, it has subgraphs with large edge density, and most importantly, *its degree distribution follows a power law*. Here, for an integer $k \leq n$, let $P(k)$ be the fraction of nodes having degree exactly $k$. If $P(k) \sim k^{-\beta}$, we say that the degree distribution of the graph follows a power law. In this case, $\beta$ is called the *power-law exponent*.

One promising model for scale-free real-world networks is the hyperbolic random graph model. A hyperbolic random graph is constructed from two parameters. First, points in the hyperbolic plane are chosen from a certain distribution depending on the parameters. Then


we consider the points as the vertices of the constructed hyperbolic random graph. For two vertices whose distance is at most a certain threshold, we add an edge between them. For illustration, see Figure 1. It is known that the degree distribution of a hyperbolic random graph follows a power-law [15]. Moreover, its diameter is small with high probability [14, 16], and it has a giant connected component [10, 17]. Including these results, the structural properties of hyperbolic random graphs have been studied extensively [11]. However, only a few algorithmic results are known. In other words, the previous work focuses on *why* we can use hyperbolic random graphs as a promising model, but only a few work focuses on *how* to use this model for solving real-world problems. We focus on the latter type of problems.

In this paper, we focus on the maximum clique problem for hyperbolic random graphs from theoretical and practical point of view. The maximum clique problem asks for a maximum-cardinality set of pairwise adjacent vertices. For general graphs, this problem is NP-hard. Moreover, it is W[1]-hard when it is parameterized by the solution size, and it is APX-hard even for cubic graphs [2]. Therefore, the theoretical study on the clique problem focuses on special classes of graphs. In fact, this problem can be solved in polynomial time for special classes of graphs such as planar graphs, unit disk graphs and hyperbolic random graphs [7, 12, 13]. More specifically, the algorithm by Bläsius et al. [7] for computing a maximum clique of a hyperbolic random graph takes $O(mn^{2.5})$ *worst-case* time. The randomness in the choice of vertices is not considered in the analysis of the algorithm. One natural question here is to design an algorithm for this problem with improved *expected* time.

To analyze our algorithm, we use the *average-case analysis*. A traditional modeling choice in the analysis of algorithms is *worst-case analysis*, where the performance of an algorithm is measured by the worst performance over all possible inputs. Although it is a useful framework in the analysis of algorithms, it does not take into account the distribution of inputs that an algorithm is likely to encounter in practice. It is possible that an algorithm performs well on most inputs, but poorly on a small number of inputs that are rarely encountered in practice. In this case, the worst-case analysis can mislead the analysis of algorithms. The field of "beyond worst-case analysis" studies ways for overcoming these limitations [25]. One simple technique studied in this field is the average-case analysis. As the hyperbolic random graph model intrinsically defines an input distribution, the average-case analysis is a natural model for analyzing algorithms for hyperbolic random graphs.

**Previous Work.** While the structural properties of hyperbolic random graphs has been studied extensively, only a few algorithmic problems have been studied. The most extensively studied algorithmic problem on hyperbolic random graphs is the generation problem: Given parameters, the goal is to generate a hyperbolic random graph efficiently. The best-known algorithms run in expected linear time [5] and in worst-case subquadratic time [26]. Also, Bläsius et al. [8] studied the problem of embedding a scale-free network into the hyperbolic plane and presented heuristics for this problem.

Very recently, classical algorithmic problems such as shortest path problems, the maximum clique problem and the independent set problem have been studied. These problems can be solved significantly faster in hyperbolic random graphs. More specifically, given a hyperbolic random graph, the shortest path between any two vertices can be computed in sublinear expected time [4, 9]. A hyperbolic random graph admits a sublinear-sized balanced separator with high probability [6]. As applications, Bläsius et al. [6] showed that the independent set problem admits a PTAS for hyperbolic random graphs, and the maximum matching problem admits a subquadratic-time algorithm. Also, the clique problem can be solved in polynomial time for hyperbolic random graphs in the worst case [7].

The clique problem has been studied extensively because it has numerous applications in various field such as community search in social networks, team formation in expert networks, gene expression and motif discovery in bioinformatics and anomaly detection in complex networks [19]. From a theoretical perspective, the best-known exact algorithm runs in $2^{0.276n}$ time in [23]. However, it is not sufficiently fast for massive real-world networks, leading to the proposal of lots of exact algorithms and heuristics for this problem on real-world networks [1, 19, 21, 24]. While these algorithms and heuristics work efficiently in practice, there is no theoretical guarantee of their efficiency.

**Our results.** We present algorithms for computing a maximum clique in a hyperbolic random graph and analyze their performances theoretically and empirically.

Given a hyperbolic random graph with parameters $\alpha$ and $C$ together with its geometric representation, we can compute a maximum clique in $O(m+n^{4.5(1-\alpha)})$ expected time, where $n$ and $m$ denotes the numbers of vertices and edges of the given graph. Here, we have $1/2 < \alpha < 1$, and the O-notation hides a constant depending on $\alpha$ and $C$. With high probability, our algorithm outperforms the previously best-known algorithm by Bläsius et al. [7] running in $O(mn^{2.5})$ time. In the case that a geometric representation is not given, our algorithm runs in $O(m+n^{6(1-\alpha)})$ expected time. This is the first algorithms for the maximum clique problem on hyperbolic random graphs not using geometric representations.

Also, we implemented our algorithms and analyzed it empirically. We run our algorithms on both synthetic data (hyperbolic random graphs) and real-world data. For hyperbolic random graphs, since it is proved that our algorithm computes a maximum clique correctly, we focus on the efficiency of the algorithms. We observed that our algorithms perform efficiently; it takes about 100ms for $n = 10^6$. For real-world networks, our algorithm gives a lower bound on the optimal solution. We observed that our algorithm performs well especially for collaboration networks and web networks. These are typical scale-free real-world networks.

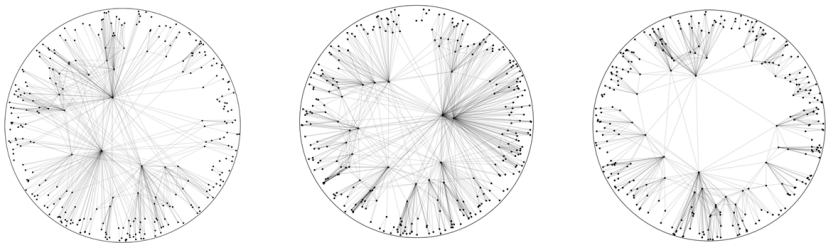
## 2 Preliminaries

Let $\mathbb{H}^2$ be the hyperbolic plane with curvature $-1$. We can handle hyperbolic planes with arbitrary (negative) curvatures by rescaling other model parameters which will be defined later. Thus it suffices to deal with the hyperbolic plane with curvature $-1$. Since the hyperbolic plane is isotropic, we choose an arbitrary point $o$ and consider it as the origin of $\mathbb{H}^2$. Also, we fix a half-line $\ell_o$ from $o$ going towards an arbitrary point, say $w$, as the *axis*. Then we can represent a point $v$ of $\mathbb{H}^2$ as $(r_v, \phi_v)$ where $r_v$ is the hyperbolic distance between $v$ and $o$, and $\phi_v$ is the angle from $\ell_o$ to the half-line from $o$ going towards $v$. We call $r_v$ and $\phi_v$ the *radial* and *angular coordinates* of $v$.

For any two points $x$ and $y$ in $\mathbb{H}^2$, we use $d(x, y)$ to denote the distance in $\mathbb{H}^2$ between $x$ and $y$. Then we have the following.

$$\begin{aligned} d(u,v) &= \cosh^{-1}(\cosh(r_u)\cosh(r_v) - \sinh(r_u)\sinh(r_v)\cos(\Delta\phi_{u,v}) \\ &\leq \cosh^{-1}(\cosh(r_u)\cosh(r_v) + \sinh(r_u)\sinh(r_v)), \end{aligned}$$

where $\Delta\phi_{u,v}$ denotes the small relative angle between $v$ and $u$ [3]. For a point $x \in \mathbb{H}^2$ and a value $r \geq 0$, we use $B_x(r)$ to denote the disk centered at $x$ with radius $r$. That is, $B_x(r) = \{v \in \mathbb{H}^2 \mid d(v, x) \leq r\}$.



(a) $\alpha = 0.75$ and $C = 3$ (b) $\alpha = 0.75$ and $C = 1$ (c) $\alpha = 0.85$ and $C = 1$

**Figure 1** HRGs with different parameters. As $C$ gets larger, the average degree gets larger (See (a–b)). As $\alpha$ gets larger, the points gets closer to the boundary of $D$ (See (b–c)).

## 2.1 Hyperbolic Random Graphs

A *hyperbolic unit disk graph (HRG)* is a graph whose vertices are placed on $\mathbb{H}^2$, and two vertices are connected by an edge if the distance between them on $\mathbb{H}^2$ is at most some threshold. This threshold is called the *radius threshold* of the graph. This is the same as the unit disk graph except that the hyperbolic unit disk graph is defined on $\mathbb{H}^2$ while the unit disk graph is defined on the Euclidean plane.

In this paper, we focus on the hyperbolic *random* graph model introduced by Papadopoulos et al. [20]. It is a family $\{\mathcal{G}_{n,\alpha,C}\}$ of distributions, indexed by the number $n$ of vertices, a parameter $C$ adjusting the average degree of a graph, and a parameter $\alpha$ determining the power-law exponent. A sample from $\{\mathcal{G}_{n,\alpha,C}\}$ is a hyperbolic unit disk graph on $n$ points (vertices) chosen independently as follows. Let $D$ be the disk centered at $o$ with radius $R = 2\ln n + C$. To pick a point $v$ in $D$, we first sample its radius $r_v$, and then sample its angular coordinate $\phi_v$. The probability density $\rho(r)$ for the radial coordinate $r_v$ is defined as

$$\rho(r) = \frac{\alpha \sinh(\alpha r)}{\cosh(\alpha R) - 1}. \tag{1}$$

Then the angular coordinate is sampled uniformly from $[0, 2\pi)$. In this way, we can sample one vertex with respect to parameters $\alpha$ and $C$, and by choosing $n$ vertices independently and by computing the hyperbolic unit disk graph with radius threshold $R$ on the $n$ vertices, we can obtain a sample from the distribution $\{\mathcal{G}_{n,\alpha,C}\}$.

An intuition behind the definition of $\rho(r)$ is as follows. To choose a point $v$ uniformly at random in $D$, we first choose its angular coordinate uniformly at random in $[0, 2\pi)$ as we did for $\mathcal{G}_{n,\alpha,C}$, and choose its radial coordinate according to the distribution with density function $\rho(r) = \frac{\sinh(r)}{\cosh(\alpha R)-1}$. But in this case, the power-law exponent of a graph is fixed. To add the flexibility to the model, the authors of [20] introduced a parameter $\alpha$ and defined the density function as in (1). Here, For $\alpha < 1$, this favors points closer to the center of $D$, while for $\alpha > 1$, this favors points closer to the boundary of $D$. For $\alpha = 1$, this corresponds to the uniform distribution [15]. For illustration, see Figure 1.

**Properties of HRGs.** Let $\mu(S)$ be the probability measure of a set $S \subseteq D$, that is,

$$\mu(S) = \int_{x \in S} \frac{\alpha \sinh(\alpha r_x)}{2\pi \cosh(\alpha R) - 1} dx.$$

For a vertex $v$ of a hyperbolic random graph $G$ with $n$ vertices, the expected degree of $v$ in $G$ is $n \cdot \mu(B_v(R) \cap D)$ by construction. Moreover, notice that $\mu(B_v(R) \cap D) = \mu(B_{v'}(R) \cap D)$ for any two vertices $v, v'$ with $r_v = r_{v'}$. Thus to make the description easier, we let $B_r(r')$ denote $B_{(r,0)}(r')$ if it is clear from the context. Note that $D = B_0(R)$.

Hyperbolic random graphs have all properties for being considered as scale-free networks mentioned above. In particular, hyperbolic random graphs with parameter $\alpha$ have power-law exponent $\beta$, where $\beta = 2\alpha + 1$ if $\alpha \geq 1/2$, and $\beta = 2$ otherwise. Most real-world networks have a power-law exponent larger than two. Thus we assume that $\alpha > 1/2$ in the paper.

## 2.2 Algorithms for the Maximum Clique Problem

In this section, we review the algorithm for this problem on hyperbolic random graphs described in [7], which is an extension of [12]. This algorithms requires geometric representations of hyperbolic random graphs. Let $G = (V, E)$ be a hyperbolic random graph.

For $\alpha \geq 1$, they showed that a hyperbolic random graph has $O(n)$ maximal cliques with high probability. Therefore, a maximum clique can be computed in linear time with high probability by just enumerating all the maximal cliques.

For $\frac{1}{2} < \alpha < 1$, they showed that the algorithm in [12] can be extended to hyperbolic random graphs. Assume first that, for a maximum clique $K$, we have two vertices $u$ and $v$ of $K$ with maximum $r = d(u, v)$. Then all vertices in $K$ are contained in the region $R_{uv} = B_u(r) \cap B_v(r)$. Then we can compute $K$ by considering the vertices in $R_{uv}$ as follows. We partition $R_{uv}$ into $R^1_{uv}$ and $R^2_{uv}$ with respect to the line through $u$ and $v$. They showed that the diameter of $R^1_{uv}$ (and $R^2_{uv}$) is at most one, and thus $V \cap R^1_{uv}$ (and $V \cap R^2_{uv}$)) forms a clique. Therefore, the subgraph $G_{uv}$ of $G$ induced by $V \cap R_{uv}$ is the complement of a bipartite graph with bipartition $(V \cap R^1_{uv}, V \cap R^2_{uv})$. Moreover, $K$ is an independent set of the complement of $G_{uv}$. Therefore, it suffices to compute an independent set of the complement of $G_{uv}$, and we can do this in in $O(n^{2.5})$ time using the Hopcroft-Karp algorithm. However, we are not given the edge $uv$ in advance. Thus we apply this procedure for every edge $uv$ of $G$, and then take the largest clique as a solution. This takes $\mathcal{O}(mn^{2.5})$ time in total.

Throughout this paper, we use $\mathbb{P}[A]$ to denote the probability that an event $A$ occurs. For a random variable $X$, we use $\mathbb{E}[X]$ to denote the expected value of $X$. Due to lack of space, some proofs and details are omitted. Missing proofs and details can be found in the full version of this paper.

# 3 Efficient Algorithm for the Maximum Clique Problem

In this section, we present an algorithm for the maximum clique problem on a hyperbolic random graph drawn from $\mathcal{R}_{n,\alpha,C}$ running in $\mathcal{O}\left(m + n^{4.5(1-\alpha)}\right)$ expected time. This algorithm correctly works for any hyperbolic unit disk graph, but its time bound is guaranteed only for hyperbolic random graphs. As we only deal with the case that $1/2 < \alpha < 1$, this algorithm is significantly faster than the algorithm in [7].

A main observation is the following. A clique of size $k$ consists of vertices of degree at least $k - 1$. That is, to find a clique of size at least $k$, removing vertices with degree less than $k - 1$ does not affect the solution. Thus once we have a lower bound, say $k$, on the size of a maximum clique, we can remove all vertices of degree less than $k$. Our strategy is to construct a sufficiently large clique (which is not necessarily maximum) as a preprocessing step so that we can remove a sufficiently large number of vertices of small degree. After applying a preprocessing step, we will see that the number of vertices we have decreases to $O(n^{1-\alpha})$ with high probability. Then we apply the algorithm in [7] to the resulting graph.

## 3.1 Computing a Sufficiently Large Clique Efficiently

In this section, we show how to compute a clique of size $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$. The algorithm is simple: Scan the vertices in the non-increasing order of their degrees, and maintain a clique $Q$, which is initially set as $\emptyset$. If the next vertex can be added to $Q$ to form a larger clique, then add it, otherwise exclude it. We can sort the vertices with respect to their degrees in $O(n+m)$ time using counting sort. Also, we can construct the clique $Q$ in $O(n+m)$ time. In the following, we call $Q$ the *initial clique*.

Now, we show that the size of the initial clique is $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$. First, we show that a sufficient large clique can be found by collecting all vertices in $B_0(R/2)$ with high probability. in Lemma 1. For its proof, see the full version.

▶ **Lemma 1.** *For any constant $c' > 0$, the vertices in $B_0(R/2-c')$ forms a clique of size $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$.*

Thus, we can get desired clique by choosing $\Omega(n^{1-\alpha})$ vertices in $B_0(R/2)$ with high probability. However, as we scan the vertices in the decreasing order of their degrees, this does not immediately imply that the size of the initial clique is $\Omega(n^{1-\alpha})$. In the following lemma, we show that the initial clique has the claimed size by showing that the $\Omega(n^{1-\alpha})$ vertices with highest degrees are contained in $B_0(R/2)$ with high probability.

▶ **Lemma 2.** *The initial clique has size $\Omega(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$.*

**Proof.** Here, we give a brief sketch of the proof. For details, see the full version. First, we show that no vertex lying outside of $B_0(R/2)$ has degree greater than $2ec_1\sqrt{n}$ with high probability for some constant $c_1$ depending only on $C, \alpha$. Then we show that no vertex in $B_0(R/2-c_2)$ has degree smaller than $2ec_1\sqrt{n}$ with high probability for some constant $c_2$. To construct the initial solution, we scan the vertices in the decreasing order of their degrees. Therefore, with probability $1-2^{-\Omega(\sqrt{n})}$, we consider all vertices in $B_0(R/2-c_2)$ before considering any vertex lying outside of $B_0(R/2)$. Therefore, the initial clique contains all vertices in $B_0(R/2-c_2)$ with high probability. By Lemma 1, the initial clique has size at least $\Omega(n^{1-\alpha})$ with high probability. ◀

## 3.2 Removing All Vertices of Small Degree

In this section, we show how to remove a sufficiently large number of vertices, and show that the size of the remaining graph is $O(n^{1-\alpha})$ with high probability. This algorithm is also simple: given the initial clique of size $k$, we repeatedly delete all vertices of degree smaller than $k$. We call the resulting graph the *kernel*. Then no vertex in the kernel has degree smaller than $k$ at the end of the process. This process can be implemented in linear time as follows: maintain the queue of vertices of degree smaller than $k$, and maintain the degree of each vertex. Then remove the vertices in the queue in order. Whenever a vertex $v$ is removed, update the degree of each neighbor $w$ of $v$ and insert $w$ to the queue if its degree gets smaller than $k$.

We show that the kernel has size $O(n^{1-\alpha})$. Notice that we do not specify the order of vertices we consider during the deletion process. Fortunately, the kernel size remains the same regardless of the choice of deletion ordering. A proof of Lemma 3 can be found in the full version of this paper.

▶ **Lemma 3.** *In any order of deleting vertices, we can get a unique kernel.*

Because of the uniqueness of the kernel, for analysis, we may fix a specific deletion ordering and slightly modify the deletion process as follows. Imagine that we scan the vertices in the decreasing order of their radial coordinates. If the degree of a current vertex (in the remaining graph) is at least the size of the initial clique size, then we terminate the deletion process. Otherwise, we delete the current vertex, and consider the next vertex. By Lemma 3, the number of remaining vertices is at least the size of the kernel. In the following lemma, we analyze the number of remaining vertices.

▶ **Lemma 4.** *Given an initial solution of size $\Omega(n^{1-\alpha})$, then the size of the kernel is $O(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$.*

**Proof.** Here, we give a sketch of the proof only. For details, see the full version.

Let $K$ be the initial solution, and let $r'$ be a value such that $n\mu(B_0(r')\cap B_{r'}(R))$ is $\frac{|K|}{2e}$. Then we show that all the vertices with radial coordinates larger than $r'$ are removed with probability $1-2^{-\Omega(n^{1-\alpha})}$. To do this, for a vertex $v$, we define the *inner degree* of $v$ as the number of its neighboring vertices whose radial coordinates are smaller than the radial coordinate of $v$. The expected inner degree of a vertex with radial coordinate $r \geq r'$ is at most $n\mu(B_0(r')\cap B_{r'}(R)) = \frac{|K|}{2e}$.

Chernoff bound implies that for a vertex with radial coordinate larger than $r'$, the probability that its inner degree is larger than the size of the initial clique is at most $2^{-|K|}$. By the union bound over at most $n$ vertices with radial coordinates larger than $r'$, the probability that no vertex with radial coordinate larger than $r'$ has inner degree larger than than the size of the initial clique is at most $n2^{-|K|} = 2^{-\Omega(n^{1-\alpha})}$. In other words, with probability $1-n2^{-c_3 n^{1-\alpha}} = 1-2^{-\Omega(n^{1-\alpha})}$, all vertices with radial coordinates larger than $r'$ have inner degree larger than the size of the initial clique.

If this event happens, we remove all vertices with coordinates larger than $r'$. We show that the number of vertices with coordinates at most $r'$ is at most $O(n^{1-\alpha})$ with probability $1-2^{-\Omega(n^{1-\alpha})}$. Therefore, the probability that the number of remaining vertices after applying the deletion process is at most $O(n^{1-\alpha})$ is at least $1-2^{-\Omega(n^{1-\alpha})}$. ◀

Although the deletion process we use for analysis requires the geometric representation of $G$, the original deletion process does not require the geometric representation of $G$. By combining the argument in Section 3.1 and Section 3.2, we have the following theorem.

▶ **Theorem 5.** *Given a graph drawn from $\mathcal{G}_{n,\alpha,C}$ with $\frac{1}{2} < \alpha < 1$ and its geometric representation, we can compute its maximum clique in $\mathcal{O}\left(m+n^{4.5(1-\alpha)}\right)$ expected time.*

## 4 Efficient Robust Algorithm for the Maximum Clique Problem

In this section, we present the first algorithm for the maximum clique problem on hyperbolic random graphs which does not require geometric representations. In many cases, a geometric representation of a graph is not given. In particular, real-world networks such as social and collaboration networks are not defined based on geometry although they share properties with HRGs. As we want to use hyperbolic random graphs as a model for such real-world networks, it is necessary to design algorithms not requiring geometric representations.

Our main key tool in this section is the notion of *co-bipartite neighborhood edge elimination ordering* (CNEEO) introduced by Raghavan and Spinrad [22]. It can be considered as a variant of a perfect elimination ordering. Let $G$ be an undirected graph. Let $L=(e_1,e_2,\dots e_m)$ be an edge ordering of all edges of $G$. Let $G_L[i]$ be the subgraph of $G$ with the edge set $\{e_i,e_{i+1}\dots e_m\}$. For a vertex $v\in V$, let $N_G(v)$ denote the set of neighbors of $v$ in $G$. Then

$L$ is called a *co-bipartite neighborhood edge elimination ordering* (CNEEO) if for each edge $e_i = (u_i, v_i)$, the subgraph of $G$ induced by $N_{G_L[i]}(u_i) \cap N_{G_L[i]}(v_i)$ is co-bipartite. Here, a *co-bipartite graph* is a graph whose complement is bipartite.

Raghavan and Spinrad [22] presented an algorithm for computing a CNEEO in polynomial time if it exists. It uses a simple greedy algorithm: compute the edges of a CNEEO one by one, and add an edge immediately if it does not violate the condition of a CNEEO. Moreover, they presented a polynomial-time algorithm for computing a maximum clique in polynomial time assuming a CNEEO is given. Since Raghavan and Spinrad [22] did not give an explicit time bound on their algorithm, we analyzed their algorithm and confirmed that their algorithm takes $O(m^3 + mn^{2.5})$ time in the full version. Note that this time bound holds for an arbitrary graph (not necessarily a hyperbolic unit disk graph) admitting a CNEEO.

We show that a hyperbolic unit disk graph admits a CNEEO. This immediately leads to an $O(m^3 + mn^{2.5})$-time algorithm for the maximum clique problem. Its proof can be found in the full version. In the case of hyperbolic *random* graphs, we can solve the problem even faster. As we did in Section 3, we compute an initial clique, remove vertices of small degrees, and then obtain a kernel of small size. Recall that these procedures do not require geometric representations. Note that the kernel is also a hyperbolic unit disk graph because we remove vertices only. The number of vertices of the kernel is $O(n^{1-\alpha})$ and the edges is $O(n^{2-2\alpha})$ in probability $1 - 2^{-\Omega(n^{1-\alpha})}$. Thus, we have the following theorem.

▶ **Corollary 6.** *Given a graph drawn from $\mathcal{G}_{n,\alpha,C}$ with $\frac{1}{2} < \alpha < 1$, we can compute a maximum clique in $\mathcal{O}\left(m + n^{6(1-\alpha)}\right)$ expected time without its geometric representation.*

**Heuristics for real-world networks.** A main motivation of the study of hyperbolic random graphs is to obtain heuristics for analyzing real-world networks. Many real-world networks share salient features with hyperbolic random graphs, but this does not mean that many real-world networks *are* hyperbolic random graphs. Because the algorithm in Corollary 6 is aborted for a graph not admitting a CNEEO, one cannot expect that this algorithm works correctly for many real-world networks. In fact, only a few real-world networks admit CNEEO as we will see in Section 6. That is, for most of real-world networks, the algorithm in Corollary 6 is aborted.

However, in this case, we can obtain a lower bound on the optimal clique sizes, and moreover, we can reduce the size of the graph. Although we do not have any theoretical bound here, our experiments showed that the size of the clique we can obtain is close to the optimal value for many instances. For details, see the full version.

## 5 Improving Performance through Additional Optimizations

For implementation, we introduce the following two minor techniques for improving the performance of the algorithms. Although these techniques do not improve the performance theoretically, they improve the performance empirically. Recall that our algorithms consist of two phases: Computing a kernel of size $O(n^{1-\alpha})$, and then computing a maximum clique of the kernel. As the first phase can be implemented efficiently, we focus on the second phase here. Again, the second phase has two steps. With geometric representations, we first compute a CNEEO, and then compute a maximum clique using the CNEEO. Without geometric representations, we consider every edge, and then compute a maximum clique in the subgraph induced by the common neighbors of the endpoint of the edge. The first technique applies to both of the two steps, and the second technique applies to the first step.

## 5.1 Skipping Vertices with Low Degree

The main observation of our kernelization algorithm is that, for any lower bound $k$ on the size of a maximum clique, a vertex of degree less than $k$ does not participate in a maximum clique. The first technique we use in the implementation is to make use of this observation also for computing a CNEEO, and for computing a maximum clique using the CNEEO.

While computing a CNEEO, the lower bound $k$ we have does not change; it is the size of the initial clique. Whenever we access a vertex, we check if its degree is less than $k$. If so, we remove this vertex from the kernel, and do not consider it any more. One can consider this as "lazy deletion." Then once we have a CNEEO, we scan the edges in the CNEEO, and for each edge, we compute a maximum clique of the subgraph defined by the edge. If it is larger than the lower bound $k$ we have, we update $k$ accordingly. In this process, whenever we find a vertex of degree less than $k$, we remove it immediately. Moreover, if the subgraph defined by each edge of CNEEO has vertices less than $k$, we skip this subgraph as it does not contain a clique of size larger than $k$.

## 5.2 Introducing the Priority of Edges

Recall that the second phases consists of two steps: computing a CNEEO, and computing a maximum clique using the CNEEO. In this section, we focus on the first step.

**With Geometric Representations.** In this case, we use the $O(m'n'^{2.5})$-time algorithm by [7] for computing a maximum clique of the kernel with $n'$ vertices and $m'$ edges. Although it is the theoretically best-known algorithm, we observed that computing a maximum clique using a CNEEO is more efficient practically. By the proof in the full version, the list of the edges sorted in the non-decreasing order of their lengths is a CNEEO. Without using a CNEEO, for each edge $uv$, we have to compute the subgraph of $G$ induced by the common neighbors of $u$ and $v$ in $G$. On the other hand, once we have a CNEEO, it suffices to consider the subgraph of $G$ induced by the common neighbors of $u$ and $v$ in $G_L$, where $G_L$ is the subgraph of $G$ with the edges coming after $uv$ in the CNEEO. If $uv$ lies close to the last edge in the CNEEO, the number of common neighbors of $u$ and $v$ in $G_L$ can be significantly smaller than the number of their common neighbors in $G$. This can lead to the performance improvement.

**Without Geometric Representations.** In this case, we compute a CNEEO in a greedy fashion. Starting from the empty sequence, we add the edges one by one in order. For each edge $e$ not added to the current ordering, we check if the common neighbors of the endpoints of $e$ in the kernel is co-bipartite. If an edge passes this test, we add it to the ordering. It is time-consuming especially when only a few edges can pass the test. To avoid considering the same edge repeatedly, we use the following observation. Once an edge $e$ fails this test, it cannot pass the test unless one of its incident edges are added to the ordering. Using this observation, we classify the edges into two sets: active edges and inactive edges. In each iteration, we consider the active edges only. Once an edge fails the test, then it becomes inactive. Once an edge passes the test, we make all its incident edges active. In this way, we can significantly improve the running time especially for graphs that do not accept CNEEO.

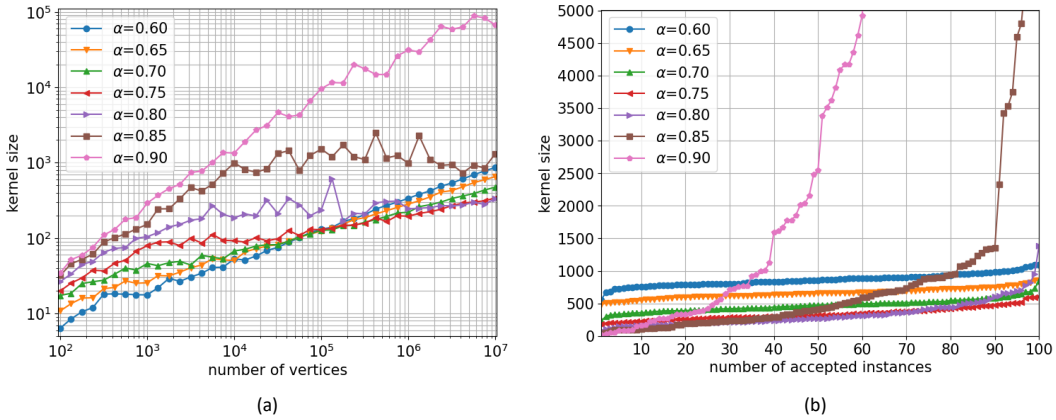



**Figure 2** (a) Comparison of the kernel sizes with varying $\alpha$. Here, $\delta = 10$. (b) Cactus plot of the kernel size versus the number of accepted instances for each value of $\alpha$ with fixed $n = 10^7$.

## 6 Experimental Evaluation

In this section, we evaluate the performance of our algorithm mainly on hyperbolic random graphs and real-world networks.

**Environment and data.** We implemented our algorithm using C++17. The code were compiled with GNU GCC version 11.3.0 with optimization flag "-O2". All tests were run on a desktop with Rygen 7 3800X CPU, 32GB memory, and Ubuntu 22.04LTS.

We evaluate the performance of our algorithm on hyperbolic random graphs and real-world networks. For hyperbolic random graphs, we generate graphs using the open source library GIRGs [5] by setting parameters differently. Recall that we have three parameters $n$, $C$ and $\alpha$. Here, instead of $C$, we use the average degree, denoted by $\delta$, as a parameter because $\delta$ can be represented as a function of $C$ and $\alpha$. As we consider the average performance of our algorithm, we sampled 100 random graphs for fixed parameters $n, \delta$ and $\alpha$, and then calculate the average results (the size of kernels or the running times).

For real-world networks, we use the SNAP dataset [18]. It contains directed graphs and non-simple graphs as well. In this case, we simply ignore the directions of the edges and interpret all directed graphs as undirected graphs. Also, we collapse all multiple edges into a single edge and remove all loops.

### 6.1 Experiment on Hyperbolic Random Graphs: Kernel Size

We showed that the size of the kernel of the hyperbolic random graph is $O(n^{1-\alpha})$ with probability $1 - 2^{\Omega(n^{1-\alpha})}$. In this section, we evaluated the tendency on the size of the kernel experimentally as $n$, $\alpha$ and $\delta$ change. Here, $\alpha$ controls the power-law exponent, and $\delta$ is the average degree of the graph. For experiments for $\delta$, see the full version.

Figure 2 shows the tendency on the size of the kernel as $\alpha$ changes. Here, we fix $\delta = 10$. Figure 2(a) shows a plot of the kernel size versus the number of vertices of a graph on a log–log scale for each value of $\alpha$. We generate 100 instances randomly and take the average of their results for each point in the plot. Figure 2(b) shows a cactus plot of the kernel size versus the number of accepted instances for each value of $\alpha$. Here, for a fixed kernel size $k$, an instance is said to *accepted* if our algorithm returns a kernel of size at most $k$ for this instance. Here, we fix $n = 10^7$ and $\delta = 10$.

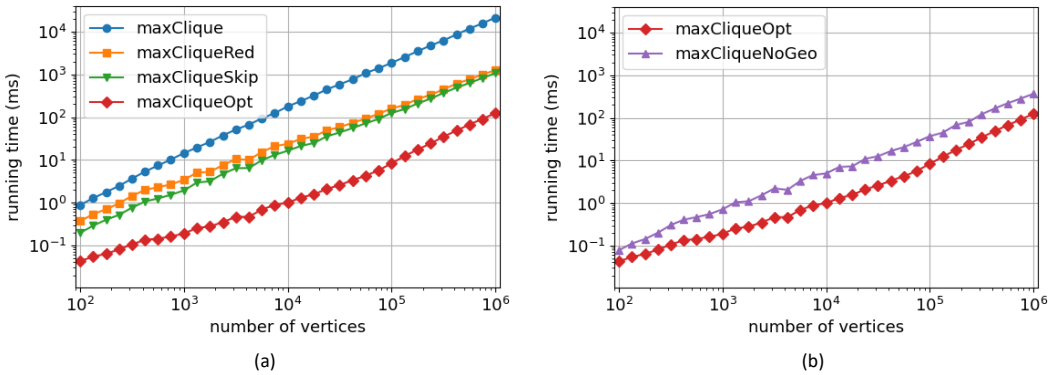



**Figure 3** (a) Comparison of running times of different versions of our algorithm. (b) Comparison of running times with and without geometric representations. Here, $\alpha = 0.75$ and $\delta = 10$.

**Table 1** Running time for each operation. The unit of time is a millisecond.

| | INIT | KERNEL | CNEEO | CONST | INDEP | OTHER | TOTAL |
|---|---|---|---|---|---|---|---|
| `MaxClique` | - | - | - | 21 516.34 | 4 470.38 | 8.18 | 25 994.90 |
| `MaxCliqueRed` | 15.67 | 89.27 | - | 1 126.95 | 37.92 | 2.88 | 1 272.70 |
| `MaxCliqueSkip` | 15.59 | 88.62 | - | 925.33 | 30.77 | 2.88 | 1 063.19 |
| `MaxCliqueOpt` | 15.64 | 88.61 | 1.07 | 12.55 | 1.19 | 2.88 | 121.94 |
| `MaxCliqueNoGeo` | 15.32 | 89.25 | 258.45 | 5.82 | 0.93 | 2.96 | 372.74 |

For $\alpha \leq 0.8$, the size of the average kernel decreases for sufficiently large $n$, say $n = 10^7$, as $\alpha$ increases in Figure 2(a). Also, the kernel sizes for all instances are concentrated on the average kernel size for each $\alpha \leq 0.8$ in Figure 2(b). This is consistent with Lemma 4 stating that the kernel size is $O(n^{1-\alpha})$ with high probability. However, this fact does not hold for $\alpha > 0.8$ in Figure 2(a), and the reason for this can be seen in Figure 2(b). At $\alpha = 0.85$, approximately 10% of instances did not have kernels of size at most 1500, and at $\alpha = 0.9$, over 60% of instances did not have such kernels. Notice that the plot sharply increases when the kernel size exceeds 1500. The success probability stated in Lemma 4 is $1 - 2^{\Omega(n^{1-\alpha})}$, which decreases as $\alpha$ increases. In other words, if $n$ is not sufficiently large, it is possible that the success probability $1 - 2^{\Omega(n^{1-\alpha})}$ is not sufficiently large for $\alpha > 0.8$. That is, if we increase the number of vertices on our experiments, we would get the desired tendency on the kernel size for all values $\alpha$. Nevertheless, at $n = 10^7$, our algorithm removes a significant number of vertices, leaving only 0.01% of vertices at $\alpha = 0.85$ and only 1% at $\alpha = 0.9$.

## 6.2 Experiment on Hyperbolic Random Graphs: Running Time

In this section, we conducted experiments for evaluating the running times of different versions of our algorithms. Figure 3 shows a plot of the number of vertices versus the running time of each version of our algorithm. Here, we fix $\alpha = 0.75$ and $\delta = 10$. Each point of the plot is averaged for 100 instances. Figure 3(a) shows a plot for the algorithm, denoted by `MaxClique`, by Bläsius et al. [7] and three different versions of the algorithm using geometric representations: `MaxCliqueRed`, `MaxCliqueSkip`, and `MaxCliqueOpt`. More specifically, `MaxCliqueRed` denotes the algorithm described in Section 4. `MaxCliqueSkip` denotes the algorithm described in Section 5.1 that skips low-degree vertices. `MaxCliqueOpt` denotes the algorithm described in Section 5.2 that introduces priorities of edges. As expected, `MaxCliqueOpt` outperforms all other versions of the algorithms in this experiment.

**Table 2** The performance of our algorithm on the real-world data.

| | $\|V\|$ | $\|E\|$ | $\|V_{\text{kernel}}\|$ | $\|V_{\text{left}}\|$ | $\|E_{\text{left}}\|$ | runtime | $\omega_{\text{kernel}}$ | $\omega_{\text{eval}}$ | $\omega$ |
|---|---|---|---|---|---|---|---|---|---|
| as-skitter | 1 696,415 | 11 095,298 | 28 787 | 17 033 | 693 272 | 342.63 | 37 | $\geq 63$ | 67 |
| ca-AstroPh | 18 771 | 198 050 | 3 679 | 0 | 0 | 1.18 | 23 | 57 | 57 |
| ca-CondMat | 23 133 | 93 439 | 13 464 | 0 | 0 | 0.07 | 4 | 26 | 26 |
| ca-HepPh | 11 204 | 118 489 | 0 | 0 | 0 | 0.00 | 239 | 239 | 239 |
| com-amazon | 334 863 | 925 872 | 255 473 | 0 | 0 | 0.55 | 3 | 7 | 7 |
| com-dblp | 317 080 | 1 049 866 | 1 716 | 0 | 0 | 0.21 | 26 | 114 | 114 |
| com-lj | 3 997 962 | 34 681 189 | 1 713 237 | 126 388 | 4 587 418 | 2 316.20 | 7 | $\geq 289$ | 327 |
| com-youtube | 1 134 890 | 2 990 443 | 36 716 | 7 919 | 211 051 | 53.48 | 13 | $\geq 14$ | 17 |
| Gnutella31 | 62 586 | 147 892 | 33 816 | 0 | 0 | 0.05 | 2 | 4 | 4 |
| Slashdot0811 | 77 360 | 469 180 | 14 315 | 1 503 | 40 418 | 7.94 | 10 | $\geq 17$ | 26 |
| Slashdot0902 | 82 168 | 504 229 | 13 964 | 1 543 | 42 215 | 9.24 | 11 | $\geq 17$ | 27 |
| soc-Epinions1 | 75 879 | 405 740 | 9 337 | 3 717 | 148 354 | 26.09 | 10 | $\geq 22$ | 23 |
| soc-pokec | 1 632 803 | 22 301 964 | 1 252 317 | 54 101 | 924 531 | 288.37 | 4 | $\geq 29$ | 29 |
| web-BerkStan | 685 230 | 6 649 470 | 27 058 | 25 593 | 589 913 | 1 224.22 | 18 | $\geq 201$ | 201 |
| web-Google | 875 713 | 4 322 051 | 193 406 | 2 068 | 20 426 | 8.60 | 10 | $\geq 44$ | 44 |
| web-NotreDame | 325 729 | 1 090 108 | 51 227 | 760 | 8 181 | 4.86 | 6 | $\geq 155$ | 155 |
| web-Stanford | 281 903 | 1 992 636 | 32 123 | 10 721 | 249 272 | 177.79 | 18 | $\geq 61$ | 61 |
| WikiTalk | 2,394 385 | 4 659 563 | 70 130 | 10 421 | 520 338 | 447.69 | 7 | $\geq 16$ | 26 |
| Wiki-Vote | 7 115 | 100 762 | 2 913 | 1 802 | 62 893 | 6.34 | 9 | $\geq 13$ | 17 |

For a precise analysis, we evaluated the running time of each task for our algorithms and reported them in in Table 1. More specifically, the algorithms conduct six tasks: INIT denotes the task of finding an initial solution. KERNEL denotes the task of finding the kernel. CNEEO denotes the task of computing a CNEEO. CONST denotes the task of constructing a co-bipartite graph by considering the common neighbors of the endpoints of each edge. INDEP denotes the task of computing a maximum independent set of the complement of a co-bipartite graph. OTHER denotes all the other tasks such as the initialization for variables and caches. TOTAL denotes the entire tasks of our algorithm.

As expected, `MaxCliqueRed` outperforms `MaxClique` significantly. However, CONST is still a time-consuming task for `MaxCliqueRed`. Thus we focus on optimization techniques for CONST and provides `MaxCliqueSkip` and `MaxCliqueOpt` in Section 5. Although `MaxCliqueSkip` gives a performance improvement, it still takes a significant amount of time in CONST and INDEP. `MaxCliqueOpt` computes a CNEEO by sorting the edges with respect to their lengths. This allows us to manage degree efficiently and apply low-degree skip technique to a larger number of vertices. This significantly improves the running time of `MaxCliqueSkip` for CONST and INDEP. This algorithm runs in about 100ms even at $n = 10^6$, exhibiting a performance improvement over 200 times compared to `MaxClique`.

Next, we compared the running times of two algorithms with and without geometrical representations. In the case that a geometric representation is given, we use `MaxCliqueOpt`. If a geometric representation is not given, we use the algorithm in Section 4 and denote it by `MaxCliqueNoGeo`. In `MaxCliqueOpt`, we can quickly compute a CNEEO by sorting the edges in non-decreasing order of length. However, `MaxCliqueNoGeo` computes a CNEEO in a greedy approach which incurs significant overhead. Despite this, the performance of `MaxCliqueNoGeo` in Figure 3(b) does not show a significant difference compared to `MaxCliqueOpt`, and it even outperforms `MaxCliqueSkip`.

## 6.3 Experiment on Real-World Dataset

Our algorithm can heuristically find large cliques for real-world data. We conducted experiments on several real-world datasets and recorded these results in Table 2. The unit of the running time is a second. $|V|$ and $|E|$ denote the numbers of vertices and edges of the input graph, respectively. $|V_{\text{kernel}}|$ denotes the number of vertices of the kernel. $|V_{\text{left}}|$ and $|E_{\text{left}}|$ denote the numbers of vertices and edges of the remaining graph. Also, $\omega_{\text{kernel}}$ denotes the size of the initial clique, $\omega_{\text{eval}}$ denotes the size of the clique computed from our algorithm, and $\omega$ denotes the size of the maximum clique of the graph. Here, $\omega$ is the correct answer given by the dataset. If a given graph accepts a CNEEO, it is theoretically guaranteed that $\omega_{\text{eval}}$ is the exact solution and $|V_{\text{left}}| = |E_{\text{left}}| = 0$. Otherwise, $\omega_{\text{eval}}$ is a lower bound on the exact solution, and $G_{\text{left}} = (V_{\text{left}}, E_{\text{left}})$ has a maximum clique if $\omega_{\text{eval}}$ is strictly smaller than the exact solution.

The collaboration networks such as ca-AstroPh, CondMat, HepPh, and com-dblp are one of the well-known scale-free networks. These networks accept a CNEEO, allowing us to find the exact maximum clique. Moreover, we were able to find a CNEEO considerably faster for these networks than for other graphs in our experiments. Web graphs such as web-BerkStan, web-Google, web-Notre Dame, and web-Stanford are also one of the well-known scale-free networks. Although these graphs do not accept a CNEEO, we were able to reduce the number of vertices and edges significantly, and we obtained maximum cliques. For the other graphs we tested, we were able to obtain lower bounds that were close to the maximum clique size in most cases, and we were able to significantly reduce the size of the graphs.

# 7 Conclusion

We presented improved algorithms for the maximum clique problem on hyperbolic random graphs. Our algorithms find a sufficiently large initial solution and find a sufficiently small kernel in linear time, which greatly improves the average time complexity and practical running time. Also we gave the first algorithm for the maximum clique problem on hyperbolic random graphs without geometrical representations. Beyond the hyperbolic random graph, we applied these algorithms to real-world dataset and obtained lower bounds close to the optimum solutions for most of instances.

There are two possible directions for further improvement on our algorithms. First, we compute a maximum clique of hyperbolic random graphs using the framework for computing a maximum clique of unit disk graphs in [12]. Recently, Espenant et al. [13] improved the algorithm [12] and presented an $O(n^{2.5} \log n)$-time algorithm for the maximum clique problem on unit disk graphs. It would be interesting if this technique can be applied to hyperbolic geometry. Second, the bottleneck of our algorithm lies in constructing a CNEEO. Especially, for most of real-world dataset, most of the running time is devoted to constructing a CNEEO. Thus to speed up the overall performance, this step must be improved.

## References


1 James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics: 5th Latin American Symposium Cancun, Mexico, April 3–6, 2002 Proceedings 5*, pages 598–612. Springer, 2002.

2 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theoretical Computer Science*, 237(1-2):123–134, 2000.

3 James W Anderson. *Hyperbolic geometry*. Springer Science & Business Media, 2006.

4 Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. Efficient shortest paths in scale-free networks with underlying hyperbolic geometry. *ACM Transactions on Algorithms (TALG)*, 18(2):1–32, 2022.

5 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. Efficiently generating geometric inhomogeneous and hyperbolic random graphs. *Network Science*, 10(4):361–380, 2022.

6 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Hyperbolic random graphs: Separators and treewidth. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

7 Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. Cliques in hyperbolic random graphs. *Algorithmica*, 80(8):2324–2344, 2018.

8 Thomas Bläsius, Tobias Friedrich, Anton Krohmer, and Sören Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. *IEEE/ACM transactions on Networking*, 26(2):920–933, 2018.

9 Thomas Bläsius, Tobias Friedrich, and Christopher Weyand. Efficiently computing maximum flows in scale-free networks. In *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

10 Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. On the largest component of a hyperbolic model of complex networks. *The Electronic Journal of Combinatorics*, pages P3–24, 2015.

11 Elisabetta Candellero and Nikolaos Fountoulakis. Clustering and the hyperbolic geometry of complex networks. In *Algorithms and Models for the Web Graph: 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings 11*, pages 1–12. Springer, 2014.

12 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

13 Jared Espenant, J. Mark Keil, and Debajyoti Mondal. Finding a maximum clique in a disk graph. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.SoCG.2023.30.

14 Tobias Friedrich and Anton Krohmer. On the diameter of hyperbolic random graphs. *SIAM Journal on Discrete Mathematics*, 32(2):1314–1334, 2018.

15 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: degree sequence and clustering. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part II 39*, pages 573–585. Springer, 2012.

16 Marcos Kiwi and Dieter Mitsche. A bound for the diameter of random hyperbolic graphs. In *2015 Proceedings of the Twelfth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 26–39. SIAM, 2014.

17 Marcos Kiwi and Dieter Mitsche. On the second largest component of random hyperbolic graphs. *SIAM Journal on Discrete Mathematics*, 33(4):2200–2217, 2019. doi:10.1137/18M121201X.

18 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

19 Can Lu, Jeffrey Xu Yu, Hao Wei, and Yikai Zhang. Finding the maximum clique in massive graphs. *Proceedings of the VLDB Endowment*, 10(11):1538–1549, 2017.

20 Fragkiskos Papadopoulos, Dmitri Krioukov, Marián Boguná, and Amin Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *2010 Proceedings IEEE Infocom*, pages 1–9. IEEE, 2010.

21 Bharath Pattabiraman, Md Mostofa Ali Patwary, Assefaw H Gebremedhin, Wei-keng Liao, and Alok Choudhary. Fast algorithms for the maximum clique problem on massive sparse graphs. In *Algorithms and Models for the Web Graph: 10th International Workshop, WAW 2013, Cambridge, MA, USA, December 14-15, 2013, Proceedings 10*, pages 156–169. Springer, 2013.

**22** Vijay Raghavan and Jeremy Spinrad. Robust algorithms for restricted domains. *Journal of algorithms*, 48(1):160–172, 2003.

**23** John Michael Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7(3):425–440, 1986.

**24** Ryan A Rossi, David F Gleich, Assefaw H Gebremedhin, and Md Mostofa Ali Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 365–366, 2014.

**25** Tim Roughgarden. *Beyond the worst-case analysis of algorithms*. Cambridge University Press, 2021.

**26** Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. In *Algorithms and Computation: 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 467–478. Springer, 2015.

# Parameterized Complexity of Equality MinCSP

**George Osipov**
Department of Computer and Information Science, Linköping University, Sweden

**Magnus Wahlström**
Department of Computer Science, Royal Holloway, University of London, UK

## Abstract

We study the parameterized complexity of MinCSP for so-called *equality languages*, i.e., for finite languages over an infinite domain such as $\mathbb{N}$, where the relations are defined via first-order formulas whose only predicate is $=$. This is an important class of languages that forms the starting point of all study of infinite-domain CSPs under the commonly used approach pioneered by Bodirsky, i.e., languages defined as reducts of finitely bounded homogeneous structures. Moreover, MinCSP over equality languages forms a natural class of optimisation problems in its own right, covering such problems as Edge Multicut, Steiner Multicut and (under singleton expansion) Edge Multiway Cut. We classify MinCSP($\Gamma$) for every finite equality language $\Gamma$, under the natural parameter, as either FPT, W[1]-hard but admitting a constant-factor FPT-approximation, or not admitting a constant-factor FPT-approximation unless FPT=W[2]. In particular, we describe an FPT case that slightly generalises Multicut, and show a constant-factor FPT-approximation for Disjunctive Multicut, the generalisation of Multicut where the "cut requests" come as disjunctions over $O(1)$ individual cut requests $s_i \neq t_i$. We also consider *singleton expansions* of equality languages, enriching an equality language with the capability for assignment constraints $(x = i)$ for either a finite or infinitely many constants $i$, and fully characterize the complexity of the resulting MinCSP.

**2012 ACM Subject Classification** Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** parameterized complexity, constraint satisfaction, parameterized approximation

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.86

**Related Version** *Full Version*: https://arxiv.org/abs/2305.11131

**Funding** *George Osipov*: Supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

## 1 Introduction

Let $D$ be a fixed domain, and let $\Gamma$ be a finite set of finitary relations over $D$. $\Gamma$ is referred to as a *constraint language*. A *constraint* over $\Gamma$ is a pair $(R, X)$, less formally written $R(X)$, where $R \in \Gamma$ is a relation of some arity $r$ and $X = (x_1, \ldots, x_r)$ is a tuple of variables. It is *satisfied* by an assignment $\alpha$ if $(\alpha(x_1), \ldots, \alpha(x_r)) \in R$. For a constraint language $\Gamma$, the *constraint satisfaction problem* over $\Gamma$, CSP($\Gamma$), is the problem where an instance $I$ is a collection of constraints over $\Gamma$, on some set of variables $V$, and the question is if there is an assignment satisfying all constraints in $I$. In the optimization variant MinCSP($\Gamma$), the input also contains an integer $k$ and the question is whether there is an assignment such that all but at most $k$ constraints are satisfied. Less formally, a constraint language $\Gamma$ determines the "type of constraints" allowed in an instance of CSP($\Gamma$) or MinCSP($\Gamma$), and varying the constraint language defines problems of varying complexity (such as $k$-SAT, $k$-Colouring, $st$-Min Cut, etc.). After decades-long investigations, *dichotomy theorems* have been established for these problems: for every constraint language over a finite domain, CSP($\Gamma$) and MinCSP($\Gamma$) is either in P or NP-complete, and the characterizations are


known [16, 40, 38, 31]. For fixed cases, such as the Boolean domain $D = \{0, 1\}$, *parameterized* dichotomies are also known, characterizing every problem MinCSP($\Gamma$) as either FPT or W[1]-hard [29], and similarly for approximate FPT algorithms [12]. This work represents significant advancements of our understanding of tractable and intractable computational problems (classical or parameterized).

But as highlighted by Bodirsky [3, 9], there are also many problems from a range of application domains that do not lend themselves to a formulation in the above CSP framework, yet which can be formulated via CSPs over structures with *infinite* domains. Unfortunately, CSPs with fixed templates over infinite domains are not as well-behaved as over finite domains; it is known that the problem CSP($\Gamma$) over an infinite domain can have any computational complexity (including being intermediate), making any dichotomy impossible [6, 9]. There are also questions of how an arbitrary infinite-domain relation would be represented. The approach used by Bodirsky, which is the standard approach for the study of infinite-domain CSPs, is to consider a language $\Gamma$ as a *reduct of a finitely bounded homogeneous structure*. Less technically, consider a structure, for example $(\mathbb{Q}, <)$ or $(\mathbb{Z}, <)$, and let $\Gamma$ be a finite language where every relation in $\Gamma$ has a quantifier-free first-order definition over the structure; i.e., $\Gamma$ is a *first-order reduct* of the structure. For such languages a dichotomy is plausible, and many cases have been settled, including *temporal* CSPs, i.e., first-order reducts of $(\mathbb{Q}, <)$ [8]; *discrete temporal* CSPs, i.e., first-order reducts of $(\mathbb{Z}, <)$ [10]; CSPs over the universal random graph [11]; and many more.

Our goal is to study the parameterized complexity of MinCSPs over such structures. Many important problems in parameterized complexity, which are not well handled by CSP optimization frameworks over finite-domain CSPs, can be expressed very simply in this setting. For example, the MinCSP with domain $\mathbb{Q}$ and the single relation $<$ is equivalent to the Directed Feedback Arc Set problem, i.e., given a digraph $D$ and an integer $k$, find a set $X$ of at most $k$ arcs from $D$ such that $D - X$ is acyclic. (Here, the vertices of $D$ become variables, the arcs constraints, and the topological order of $D - X$ becomes an assignment which violates at most $|X|$ constraints.) Other examples include Subset Directed Feedback Arc Set, which corresponds to MinCSP($<, \leq$), and Symmetric Directed Multicut which corresponds to MinCSP($\leq, \neq$). The former is another important FPT problem [18], while FPT status of the latter is open [23].

The structure we study in this paper is $(\mathbb{N}, =)$. The relations definable over this structure are called *equality constraint languages*. Here, $\mathbb{N}$ is an arbitrary, countably infinite domain; first-order reducts of $(\mathbb{N}, =)$ are simply relations definable by a quantifier-free first-order formula whose only predicate is $=$. Equivalently, relations in an equality language accept or reject an assignment to their arguments purely based on the partition that the assignment induces. Since every first-order formula is allowed to use equality in this framework, equality languages are contained in *every* other class of languages studied in the framework. Hence, characterizing the complexity of equality languages is a prerequisite for studying any other structure.

Moreover, the setting also covers problems that are important in their own right, as it captures undirected *graph separation* problems. In particular, (Vertex/Edge) Multicut is defined as follows. The input is a graph $G$, an integer $k$, and a set of *cut requests* $\mathcal{T} \subseteq \binom{V(G)}{2}$, and the task is to find a set $X$ of at most $k$ vertices, respectively edges, such that for every cut request $st \in \mathcal{T}$, vertices $s$ and $t$ are in different connected components in $G - X$. Multicut is FPT parameterized by $k$ – a breakthrough result, settling a long-open question [37, 14]. As with the above examples, there appears to be no natural way of capturing Multicut as a finite-domain CSP optimization problem. However, it naturally

corresponds to MinCSP$(=, \neq)$ over domain $\mathbb{N}$, where edges correspond to soft $=$-constraints and cut requests to crisp $\neq$-constraints. Another classic problem is Multiway Cut, which is the special case of Multicut where the cut requests are $\mathcal{T} = \binom{T}{2}$ for a set $T$ of *terminal vertices* in the graph. Multiway Cut was among the first graph separation problems shown to be FPT [36], and remains a relevant problem, e.g., for the question of *polynomial kernelization* [32, 39]. While Multiway Cut is not directly captured by an equality CSP, it is captured by the *singleton expansion* of the setting, i.e., allowing "assignment constraints" of the form $x = i$ for all $i \in \mathbb{N}$.

**Related work.** Bodirsky and Kara [7] characterized CSP($\Gamma$) as in P or NP-hard for every equality language $\Gamma$. Bodirsky, Chen and Pinsker [5] characterized the structure of equality languages up to pp-definitions (*primitive positive definitions*, see Section 2); these are too coarse to preserve the parameterized complexity of a problem, but their results are very useful as a guide to our search. For much more material on CSPs over infinite domains, see Bodirsky [4]. Singleton expansions (under different names) are discussed by Bodirsky [4] and Jonsson [25]. We have taken the term from Barto et al. [1].

Many variations on cut problems have been considered, and have been particularly important in parameterized complexity [20] (see also [19]). We cover Multicut, Multiway Cut and *Steiner cut*. Given a graph $G$ and a set of terminals $T \subseteq V(G)$, a Steiner cut is an edge cut in $G$ that separates $T$, i.e., a cut $Z$ such that some pair of vertices in $T$ is disconnected in $G - Z$. Steiner Cut is the problem of finding a minimum Steiner cut. This can clearly be solved in polynomial time; in fact, using advanced methods, it can even be computed in *near-linear* time [33, 17]. Steiner Multicut is the generalization where the input contains a set $\mathcal{T} = \{T_1, \ldots, T_t\}$ of terminal sets and the task is to find a smallest-possible cut that separates every set $T_i$. Since Multiway Cut with 3 terminals is NP-hard, Steiner Multicut is NP-hard if $t \geq 3$. Bringmann et al. [15] considered parameterized variations of this and showed, among other results, that Edge Steiner Multicut is FPT even for terminal sets $T_i$ of unbounded size, if the parameter includes both $t$ and the cut size $k$. On the other hand, parameterized by $k$ alone, Steiner Multicut is W[1]-hard for terminal sets of size $|T_i| = 3$.

Other parameterized CSP dichotomies directly relevant to our work are the dichotomies for Boolean MinCSP as having constant factor FPT-approximations or being W[1]-hard to approximate [12] (with additional results in a later preprint [13]) and the recent FPT/W[1]-hardness dichotomy [29].

The area of FPT approximations has seen significant activity in recent years, especially regarding lower bounds on FPT approximations [24, 26, 2, 34]. In particular, we will need that there is no constant-factor FPT-approximation for Nearest Codeword in Boolean codes unless FPT=W[1] [13, 2], or for Hitting Set unless FPT=W[2] [34]. Lokshtanov et al. [35] considered fast FPT-approximations for problems whose FPT algorithms are slow; in particular, our result for Steiner Multicut builds on their algorithm giving an $O^*(2^{O(k)})$-time 2-approximation.

## Our Results

We study the classical and parameterized complexity of MinCSP($\Gamma$) for every finite equality language $\Gamma$, as well as for singleton expansions over equality languages. We consider both exact FPT-algorithms and constant-factor FPT-approximations, and for every finite $\Gamma$ classify whether MinCSP($\Gamma$) is in FPT, MinCSP($\Gamma$) is W[1]-hard but admits constant-factor FPT-approximation, or MinCSP($\Gamma$) is W[1]-hard to approximate within any constant. To describe the cases in more detail, we need some definitions.

Unsurprisingly, MinCSP($\Gamma$) for an equality language $\Gamma$ is NP-hard except in trivial cases, since MinCSP($=, \neq$) already corresponds to Edge Multicut. Specifically, MinCSP($\Gamma$) is in P if $\Gamma$ is *constant*, in which case every relation in $\Gamma$ contains the tuple $(1, \ldots, 1)$, or *strictly negative*, in which case every relation in $\Gamma$ contains every tuple $(1, \ldots, r)$ where all values are distinct (proper definitions of the terms are found in Section 3). In all other cases, MinCSP($\Gamma$) is NP-hard by reduction from Edge Multicut. Moreover, under the Unique Games Conjecture [27], NP-hard MinCSP($\Gamma$) does not admit polynomial-time constant-factor approximation.

To describe constraint languages $\Gamma$ that give rise to fixed-parameter tractable MinCSP($\Gamma$), let $\mathsf{NEQ}_3$ be the ternary relation which contains all tuples with three distinct values, and let a *split* constraint be a constraint $R$ of some arity $p + q$ for $p, q \geq 0$, defined (up to argument order) by

$$R(x_1, \ldots, x_p, y_1, \ldots, y_q) \equiv \bigwedge_{i,j \in [p]} (x_i = x_j) \wedge \bigwedge_{i \in [p], j \in [q]} (x_i \neq y_j).$$

We note that MinCSP($\Gamma$) with split constraints reduces to Vertex Multicut. A split constraint $R(u_1, \ldots, u_p, v_1, \ldots, v_q)$ can be represented by introducing a new vertex $c$, adding edges $cu_i$ for every $i \in [p]$ and cut requests $cv_j$ for every $i \in [q]$. However, a constraint $\mathsf{NEQ}_3(u, v, w)$ cannot be handled by a gadget. Hence, we introduce the following auxiliary graph problem, and show that it is in FPT.

| Vertex Multicut with Deletable Triples (aka Triple Multicut) | |
|---|---|
| Instance: | A graph $G$, a collection $\mathcal{T} \subseteq \binom{V(G)}{3}$ of vertex triples, and integer $k$. |
| Parameter: | $k$. |
| Question: | Are there subsets $X_V \subseteq V(G)$ and $X_\mathcal{T} \subseteq \mathcal{T}$ such that $\lvert X_V \rvert + \lvert X_\mathcal{T} \rvert \leq k$ and every connected component of $G - X_V$ intersects every triple in $\mathcal{T} \setminus X_\mathcal{T}$ in at most one vertex? |

MinCSP($\Gamma$) for $\Gamma$ with only split relations and $\mathsf{NEQ}_3$ easily reduces to Triple Multicut. To complement this result with hardness, we prove the following.

▶ **Theorem 1.** *Let $\Gamma$ be an equality constraint language that is neither constant nor strictly negative. Then MinCSP($\Gamma$) is FPT if every relation in $\Gamma$ is either split or $\mathsf{NEQ}_3$, and W[1]-hard otherwise.*

Next, we describe the cases with constant-factor FPT-approximations. Consider relation

$$R_d = (x_1 \neq y_1 \vee \cdots \vee x_d \neq y_d)$$

and let $\Gamma$ be a constraint language where every relation is defined by conjunction of relations $R_d$ and $=$. We show that MinCSP($\Gamma$) for such $\Gamma$ is constant-factor fpt-approximable, with the factor depending on $d$. Again, we introduce a new graph problem to capture this case. Let $G$ be a graph; a subset $L \subseteq \binom{V(G)}{2}$ of pairs is a *request list*, and a set of vertices $X \subseteq V(G)$ *satisfies* $L$ if there is a pair $st \in L$ separated by $X$. For a graph $G$ and a collection of request lists $\mathcal{L}$, let $\mathrm{cost}(G, \mathcal{L})$ be the minimum size of a set $X \subseteq V(G)$ that satisfies all lists in $\mathcal{L}$.

| Disjunctive Multicut | |
|---|---|
| Instance: | A graph $G$, a collection $\mathcal{L}$ of request lists, each of size at most $d$, and an integer $k$. |
| Parameter: | $k$. |
| Question: | Is $\mathrm{cost}(G, \mathcal{L}) \leq k$? |

Our main algorithmic contribution is an FPT-approximation for Disjunctive Multicut. Note that Steiner Multicut is the special case of Disjunctive Multicut where each request list is $L = \binom{T_i}{2}$ for some terminal set $T_i$. Here we obtain an improved algorithm with approximation factor 2 and running time $O^*(2^{O(k)})$.

This precisely describes the FPT-approximable cases of MinCSP($\Gamma$): For every equality constraint language $\Gamma$ such that CSP($\Gamma$) is in P, either MinCSP($\Gamma$) reduces to Disjunctive Multicut in an immediate way (up to a constant-factor approximation loss), implying a constant-factor FPT-approximation, or there is a cost-preserving reduction from Hitting Set to MinCSP($\Gamma$). We refer to the latter as MinCSP($\Gamma$) being Hitting Set-hard.

▶ **Theorem 2.** *Let $\Gamma$ be an equality constraint language such that CSP($\Gamma$) is in P. Then either MinCSP($\Gamma$) reduces to Disjunctive Multicut and has a constant-factor FPT-approximation, or MinCSP($\Gamma$) is Hitting Set-hard.*

We can summarize the main result in the following way: for every finite equality constraint language $\Gamma$, either MinCSP($\Gamma$) is FPT by reduction to Triple Multicut, MinCSP($\Gamma$) is FPT-approximable by reduction to Disjunctive Multicut, or MinCSP($\Gamma$) is Hitting Set-hard. A more technical description in terms of the allowed relations can be found in Section 3.

**Singleton Expansion.** In addition to the above (main) results, we also investigate the effect of adding constants to an equality language motivated by the problem Multiway Cut. More precisely, for an equality language $\Gamma$, we investigate the effect of adding some number of unary singleton relations $\{(i)\}$ to $\Gamma$. This is equivalent to allowing "assignment constraints" $(x = i)$ in MinCSP($\Gamma$). We consider adding either a finite number of singletons, or every singleton relation. For an equality language $\Gamma$ and an integer $c \in \mathbb{N}$, $c \geq 1$, we define $\Gamma_c^+ = \Gamma \cup \{\{(i) \mid i \in [c]\}$ as the language $\Gamma$ with $c$ different singletons added, and let $\Gamma^+$ denote $\Gamma$ with every singleton $\{(i)\}$, $i \in \mathbb{N}$ added. Edge Multiway Cut corresponds to MinCSP($\Gamma^+$) over the language $\Gamma = \{=\}$, and $s$-Edge Multiway Cut, the special case with $s$ terminals, corresponds to MinCSP($\Gamma_s^+$). By a *singleton expansion of* $\Gamma$ we refer to either the language $\Gamma' = \Gamma^+$ or $\Gamma' = \Gamma_c^+$ for some $c \in \mathbb{N}$.

As the first step of the characterization, we observe that if $\Gamma$ can express $=$ and $\neq$, then the singleton expansion adds no power, i.e., MinCSP($\Gamma^+$) reduces back to MinCSP($\Gamma$) by introducing variables $c_1, \ldots, c_m$ for arbitrarily many constants, adding constraints $c_i \neq c_j$ whenever $i \neq j$, and using constraints $x = c_i$ in place of assignments $x = i$. For the rest of the characterization, we thus study the cases that either cannot express equality, or cannot express disequality. In the former case, MinCSP($\Gamma'$) is always FPT and constant-factor approximable; the latter case is more involved. We defer the full case description to the full paper, but in summary, for any singleton expansion $\Gamma'$ of a finite equality language $\Gamma$, we characterize MinCSP($\Gamma'$) as being in P or NP-hard, being FPT or W[1]-hard, and with respect to the existence of polynomial-time or FPT constant-factor approximations. Overall, the positive cases in the characterization follow without difficulty, but completing the picture with negative results requires significant additional work, building on the structural results of Bodirsky, Chen and Pinsker [5].

**Roadmap.** Section 2 contains technical preliminaries. Section 3 gives an overview of the classification proof. Section 4 shows the FPT algorithm for Triple Multicut. Section 5 gives the FPT approximation algorithms. This version omits several proofs, the approximation result for Steiner Multicut, and the classification of CSP and MinCSP for equality constraint languages under singleton expansion. These can be found in the full version.

## 2 Preliminaries

**Graph Separation.** Let $G$ be an undirected graph. Denote the vertex set of $G$ by $V(G)$ and the edge set by $E(G)$. For a subset of edges/vertices $X$ in $G$, let $G - X$ denote the graph obtained by removing the elements of $X$ from $G$, i.e. $G - X = (V(G), E(G) \setminus X)$ if $X \subseteq E(G)$ and $G - X = G[V(G) \setminus X]$ if $X \subseteq V(G)$. A *cut request* is a pair of vertices $st \in \binom{V(G)}{2}$, and an $st$-cut/$st$-separator is a subset of edges/vertices $X$ such that $G - X$ contains no path connecting $s$ and $t$. We write that $X$ *fulfills* $st$ if $X$ is an $st$-cut/$st$-separator. We implicitly allows the inputs to cut problems such as Multiway Cut and Multicut to contain undeletable edges/vertices: for edges, with a parameter of $k$, we can include $k+1$ parallel copies; for vertices, we can replace a vertex $v$ with a clique of size $k+1$, where every member of the clique has the same neighbourhood as $v$.

**Parameterized Deletion.** A *parameterized* problem is a subset of $\Sigma^* \times \mathbb{N}$, where $\Sigma$ is the input alphabet. The parameterized complexity class FPT contains problems decidable in $f(k) \cdot n^{O(1)}$ time, where $f$ is a computable function and $n$ is the bit-size of the instance. Let $L_1, L_2 \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A mapping $F : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ is an *FPT-reduction* from $L_1$ to $L_2$ if

- $(x, k) \in L_1$ if and only if $F((x, k)) \in L_2$,
- the mapping can be computed in $f(k) \cdot n^{O(1)}$ time for some computable function $f$, and
- there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$, if $(x', k') = F((x, k))$, then $k' \leq g(k)$.

The classes W[1] and W[2] contains all problems that are FPT-reducible to Clique and Hitting Set, respectively, parameterized by the solution size. These problems are not in FPT under the standard assumptions FPT≠W[1] and FPT≠W[2]. For a thorough treatment of parameterized complexity we refer to [20].

**Constraint Satisfaction.** Fix a *domain* $D$. A relation $R$ of *arity* $r$ is a subset of tuples in $D^r$, i.e. $R \subseteq D^r$. We write $=$ and $\neq$ to denote the binary equality and disequality relations over $D$, i.e. $\{(a, b) \in D^2 : a = b\}$ and $\{(a, b) \in D^2 : a \neq b\}$, respectively. A *constraint language* $\Gamma$ is a set of relations over a domain $D$. A *constraint* is defined by a relation $R$ and a tuple of variables $\mathbf{x} = (x_1, \ldots, x_r)$, where $r$ is the arity of $R$. It is often written as $R(\mathbf{x})$ or $R(x_1, \ldots, x_r)$. An assignment $\alpha : \{x_1, \ldots, x_r\} \to D$ *satisfies* the constraint if $\alpha(\mathbf{x}) = (\alpha(x_1), \ldots, \alpha(x_r)) \in R$, and *violates* the constraint if $\alpha(\mathbf{x}) \notin R$.

| Constraint Satisfaction Problem for $\Gamma$ (CSP($\Gamma$)) | |
|---|---|
| Instance: | An instance $I$, where $V(I)$ is a set of variables and $C(I)$ is a multiset of constraints using relations from $\Gamma$. |
| Question: | Is there an assignment $\alpha : V(I) \to D$ that satisfies all constraints in $C(I)$? |

MinCSP is an optimization version of the problem seeking an assignment that minimizes the number of violated constraints. In this constraints are allowed to be *crisp* and *soft*. The *cost of assignment* $\alpha$ in an instance $I$ of CSP is infinite if it violates a crisp constraint, and equals the number of violated soft constraints otherwise. The *cost of an instance* $I$ denoted by $\text{cost}(I)$ is the minimum cost of any assignment to $I$.

| MinCSP($\Gamma$) | |
|---|---|
| Instance: | An instance $I$ of CSP($\Gamma$) and an integer $k$. |
| Question: | Is $\text{cost}(I) \leq k$? |

Next, we recall a useful notion that captures local reductions between CSPs.

▶ **Definition 3.** *Let $\Gamma$ be a constraint language over $D$ and $R \subseteq D^r$ be a relation. A* primitive positive definition (pp-definition) *of $R$ in $\Gamma$ is an instance $C_R$ of CSP$(\Gamma, =)$ with primary variables $\mathbf{x}$, auxiliary variables $\mathbf{y}$ and the following properties:*
**(1)** *if $\alpha$ satisfies $C_R$, then it satisfies $R(\mathbf{x})$,*
**(2)** *if $\alpha$ satisfies $R(\mathbf{x})$, then there exists an extension of $\alpha$ to $\mathbf{y}$ that satisfies $C_R$.*

Informally, pp-definitions can be used to simulate $R$ using the relations available in $\Gamma$ and equality: every constraint using $R$ can be replaced by a gadget based on the pp-definition, resulting in an equivalent instance. The type of reductions captured by pp-definitions is however incompatible with MinCSP because the reductions do not preserve assignment costs. This motivates the following definition.

▶ **Definition 4.** *Let $\Gamma$ be a constraint language over $D$ and $R \subseteq D^r$ be a relation. An* implementation *of $R$ in $\Gamma$ is a pp-definition of $R$ with primary variables $\mathbf{x}$, auxiliary variables $\mathbf{y}$ and an additional property: if $\alpha$ violates $R(\mathbf{x})$, there exists an extension of $\alpha$ to $\mathbf{y}$ of cost one.*

Although pp-definitions do not preserve costs, they can be used to simulate crisp constraints in MinCSP instances.

▶ **Proposition 5** (Proposition 5.2 in [30])**.** *Let $\Gamma$ be a constraint language over a domain $D$ and $R$ be a relation over $D$. Then the following hold.*
1. *If $\Gamma$ pp-defines $R$, then there is an FPT-reduction from MinCSP$(\Gamma, R)$ restricted to instances with only crisp $R$-constraints to MinCSP$(\Gamma, =)$.*
2. *If $\Gamma$ implements $R$, there is an FPT-reduction from MinCSP$(\Gamma, R)$ to MinCSP$(\Gamma, =)$.*

**Approximation.** A minimization problem over an alphabet $\Sigma$ is a triple $(\mathcal{I}, \mathrm{sol}, \mathrm{cost})$, where $\mathcal{I} \subseteq \Sigma^*$ is the set of instances, $\mathrm{sol} : \mathcal{I} \to \Sigma^*$ is a function such that maps instances $I \in \mathcal{I}$ to the sets of solutions $\mathrm{sol}(I)$, and $\mathrm{cost} : \mathcal{I} \times \Sigma^* \to \mathbb{Z}_{\geq 0}$ is a function that takes an instance $I \in \mathcal{I}$ and a solution $X \in \mathrm{sol}(I)$ as input, and returns a non-negative integer cost of the solution. Define $\mathrm{cost}(I) := \min\{\mathrm{cost}(I, X) : X \in \mathrm{cost}(I)\}$. A constant-factor approximation algorithm with factor $c \geq 1$ takes an instance $x \in \mathcal{I}$ and an integer $k \in \mathbb{N}$, and returns "yes" if $\mathrm{cost}(I) \leq k$ and "no" if $\mathrm{cost}(I) > c \cdot k$. A *cost-preserving reduction* from a problem $A = (\mathcal{I}_A, \mathrm{sol}_A, \mathrm{cost}_A)$ to $B = (\mathcal{I}_B, \mathrm{sol}_B, \mathrm{cost}_B)$ is a pair of polynomial-time computable functions $F$ and $G$ such that (1) for every $I \in \mathcal{I}_A$, we have $F(I) \in \mathcal{I}_B$ with $\mathrm{cost}_A(I) = \mathrm{cost}_B(F(I))$, and (2) for every $I \in \mathcal{I}_A$ and $Y \in \mathrm{sol}(F(I))$, we have $G(I, Y) \in \mathrm{sol}(I)$, and $\mathrm{cost}_A(I, G(I, Y)) \leq \mathrm{cost}_B(F(I), Y)$. If there is a cost-preserving reduction from $A$ to $B$, and $B$ admits a constant-factor polynomial-time/fpt approximation algorithm, then $A$ also admits a constant-factor polynomial-time/fpt approximation algorithm.

## 3 Classification Overview

We now give an overview of the complexity dichotomy. Details are deferred to the full paper. We begin with a definition of the relevant language classes. Recall that an *equality language* is a constraint language over $\mathbb{N}$ whose relations can be defined via Boolean formulas over the equality predicate. More precisely, for a set of variables $X = \{x_1, \ldots, x_n\}$, let a *positive literal* be a term $(x_i = x_j)$ and a *negative literal* a term $(x_i \neq x_j)$, $i, j \in [n]$. A *clause* is a disjunction of literals. Then every equality relation has a CNF definition as a conjunction of clauses. A relation (respectively language) is *Horn* if it (respectively every relation in the

**Table 1** Selected Horn relations $R$ and the complexity of MinCSP$(R, =, \neq)$. FPA refers to fixed-parameter approximation.

| Name | CNF Formula | Tuples | Complexity |
|---|---|---|---|
| $\mathsf{EQ}_3$ | $(x_1 = x_2) \wedge (x_2 = x_3) \wedge (x_1 = x_3)$ | $(1,1,1)$ | FPT |
| $\mathsf{NEQ}_3$ | $(x_1 \neq x_2) \wedge (x_2 \neq x_3) \wedge (x_1 \neq x_3)$ | $(1,2,3)$ | FPT |
| — | $(x_1 = x_2) \wedge (x_1 \neq x_3) \wedge (x_2 \neq x_3)$ | $(1,1,2)$ | FPT |
| $\mathsf{ODD}_3$ | $(x_1 = x_2 \vee x_1 \neq x_3) \wedge (x_1 = x_2 \vee x_2 \neq x_3) \wedge (x_1 \neq x_2 \vee x_2 \neq x_3)$ | $(1,1,1), (1,2,3)$ | Hitting Set-hard |
| $\mathsf{NAE}_3$ | $(x_1 \neq x_2 \vee x_2 \neq x_3)$ | excludes $(1,1,1)$ | W[1]-hard, FPA |
| $R^{\vee}_{\neq,\neq}$ | $(x_1 \neq x_2 \vee x_3 \neq x_4) \wedge (x_1 \neq x_3) \wedge (x_1 \neq x_4) \wedge (x_2 \neq x_3) \wedge (x_2 \neq x_4)$ | $(1,2,3,3)$, $(1,1,2,3)$, $(1,2,3,4)$ | W[1]-hard, FPA |
| $R^{\wedge}_{=,=}$ | $(x_1 = x_2) \wedge (x_3 = x_4)$ | $(1,1,1,1)$, $(1,1,2,2)$ | W[1]-hard, FPA |
| $R^{\wedge}_{\neq,\neq}$ | $(x_1 \neq x_2) \wedge (x_3 \neq x_4)$ | – *too many too list here* – | W[1]-hard, FPA |
| $R^{\wedge}_{=,\neq}$ | $(x_1 = x_2) \wedge (x_3 \neq x_4)$ | $(1,1,1,2), (1,1,2,1), (1,1,2,3)$ | W[1]-hard, FPA |

language) has a CNF definition where every clause has at most one positive literal, *negative* if positive literals only occur in singleton clauses $(x_i = x_j)$, *strictly negative* if there are no positive literals, and *conjunctive* if all clauses are singletons. Note that split relations and $\mathsf{NEQ}_3$ are both conjunctive.

Bodirsky and Kara [7] showed that for an equality language $\Gamma$, CSP$(\Gamma)$ is in P if $\Gamma$ is Horn or constant, and NP-hard otherwise. We note that MinCSP$(\Gamma)$ is trivial if $\Gamma$ is constant or strictly negative, and also show that if $\Gamma$ is Horn but not constant or strictly negative then $\Gamma$ implements the relations $=$ and $\neq$. Hence we focus on this case and assume that $\Gamma$ is Horn and $=, \neq \in \Gamma$. The polynomial-time complexity classification then follows since Edge Multicut reduces to MinCSP$(=, \neq)$. For the remaining steps, we show that

1. MinCSP$(\Gamma, =, \neq)$ admits a constant-factor FPT-approximation if $\Gamma$ is negative, otherwise it is Hitting Set-hard;
2. MinCSP$(R, =, \neq)$ is W[1]-hard if $R$ is negative but not conjunctive;
3. MinCSP$(R, =, \neq)$ is W[1]-hard if $R$ is conjunctive but neither split nor $\mathsf{NEQ}_3$;
4. MinCSP$(\Gamma, =, \neq)$ is in FPT if $\Gamma$ is conjunctive and all relations in $\Gamma$ are split or $\mathsf{NEQ}_3$.

Table 1 lists a number of Horn relations and the associated complexity of MinCSP.

Towards hardness of approximation, we recall a result of Bodirsky, Chen and Pinsker [5] that every equality language that is not negative can define $\mathsf{ODD}_3$ (see Table 1). We show that MinCSP$(\mathsf{ODD}_3, =, \neq)$ is Hitting Set-hard.

▶ **Lemma 6.** *There is a cost-preserving reduction from Hitting Set to MinCSP*$(\mathsf{ODD}_3, =, \neq)$ *where every* $\mathsf{ODD}_3$*-constraint is crisp.*

**Proof Sketch.** Let the input be $(V, \mathcal{E}, k)$, $V = \{1, \ldots, n\}$. Create an instance $(I, k)$ of MinCSP$(\mathsf{ODD}_3, =, \neq)$ starting from variables $x_1, \ldots, x_n$ and $z$, with soft constraints $x_i = z$ for all $i \in [n]$. For every set $e = \{a_1, \ldots, a_\ell\} \in \mathcal{E}$, add auxiliary variables $y_2, \ldots, y_\ell$ and crisp constraints $\mathsf{ODD}_3(x_{a_1}, x_{a_2}, y_2)$, $\mathsf{ODD}_3(y_{i-1}, x_{a_i}, y_i)$ for all $3 \leq i \leq \ell$, and $x_{a_1} \neq y_\ell$. These constraints are satisfiable if and only if not all variables $x_i$, $i \in e$ are equal, so to satisfy them it is sufficient to break a soft constraint $x_{a_i} = z$. Thus, $X \subseteq V$ is a hitting set if and only if $I - \{x_i = z : i \in X\}$ is consistent. ◀

As noted, this implies that MinCSP$(\Gamma)$ is W[1]-hard to even approximate in FPT time. Now assume that $\Gamma$ is negative. Then every relation $R \in \Gamma$ is defined by a conjunction of positive singleton clauses and strictly negative clauses. For a constant-factor approximation,

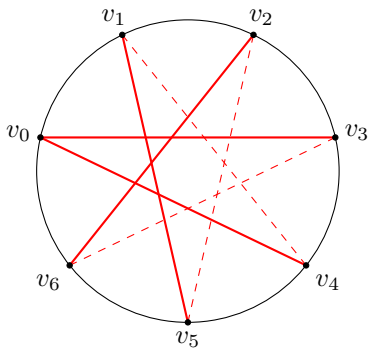



**Figure 1** An illustration of a choice gadget for $t = 3$. Black arcs correspond to equality constraints, dashed red edges to soft disequality constraints, and the bold red edge to a crisp disequality constraint.

by [12, Lemma 10] we may split these definitions into separate constraints, so we may assume that an instance of MinCSP($\Gamma$) uses constraint types $(x_i = x_j)$ and $(x_1 \neq y_1 \vee \ldots \vee x_r \neq y_r)$, of lengths $r \leq d$ for some constant $d$. Then MinCSP($\Gamma$) reduces to Disjunctive Multicut, and since $d = O(1)$ we get an FPT approximation. This settles the cases where MinCSP($\Gamma$) has a constant-factor FPT approximation.

Towards delineating FPT and W[1]-hard cases, assume that $\Gamma$ is negative but not conjunctive, and let $R \in \Gamma$ be a relation such that every CNF definition of $R$ contains a non-singleton clause. We show that in this case $\{R, =, \neq\}$ pp-defines $R^{\vee}_{\neq,\neq}$ or $\mathsf{NAE}_3$, and that MinCSP($R', =, \neq$) is W[1]-hard with crisp $R'$-constraints, where $R'$ is either $R^{\vee}_{\neq,\neq}$ or $\mathsf{NAE}_3$. The problem MinCSP($\mathsf{NAE}_3, =$) with crisp $\mathsf{NAE}_3$-constraints naturally corresponds to Steiner Multicut, which is W[1]-hard [15]. For the remaining proofs, we will need a *choice gadget*. Let $S = \{s_1, \ldots, s_t\}$ be a set. Define an instance $W(S)$ of CSP($=, \neq$) as follows. Introduce $2t + 1$ variables $v_0, \ldots, v_{2t}$ and identify indices modulo $2t + 1$, e.g. $v_0 = v_{2t+1}$. Connect variables in a cycle of equalities, i.e. add soft constraints $v_i = v_{i+1}$ for all $0 \leq i \leq 2t$. The *forward partner* of a variable $v_i$ is $f(v_i) := v_{i+t}$, i.e. the variable that is $t$ steps ahead of $v_i$ on the cycle. Add soft constraints $v_i \neq f(v_i)$ for all $0 \leq i \leq 2t$, and make the constraint $v_0 \neq v_t$ crisp. See Figure 1 for an illustration.

▶ **Lemma 7.** *Let $S$ be a set of size at least two and $W(S)$ be the choice gadget. Then* $\mathrm{cost}(W(S)) = 3$. *Moreover, if $X \subseteq W(S)$, $|X| = 3$, $W(S) - X$ is consistent and $X$ contains $v_i \neq f(v_i)$, then $X$ also contains $v_{i-1} = v_i$ and $f(v_i) = f(v_{i+1})$.*

**Proof Sketch.** Since $v_0 \neq v_t$ is a crisp constraint, we need to remove one link from the top and one from the the bottom chain of equality constraints connecting $v_0$ and $v_t$. After this deletion, at least one chain of length $\lceil \frac{2t+1-2}{2} \rceil = t$ remains, which connects a vertex $v_i$ with its forward partner $f(v_i)$. Thus, we either need to disconnect $v_i$ and $f(v_i)$, or remove the constraint $v_i \neq f(v_i)$. Observe that it is sufficient to delete $v_{i-1} = v_i$, $f(v_i) = f(v_{i+1})$ and $v_i \neq f(v_i)$ to satisfy $W(S)$. Moreover, if a deletion set $X$ of size 3 contains $v_i \neq f(v_i)$, the only pair of vertices $v_j$ and $f(v_j)$ that may remain connected in $W(S) - X$ is the pair with $j = i$. Out of the two compatible choices, only deleting $v_{i-1} = v_i$ and $f(v_i) = f(v_{i+1})$ leaves no constraint unsatisfied. ◀

Intuitively, deleting $v_{i-1} = v_i$, $f(v_i) = f(v_{i+1})$ and $v_i \neq f(v_i)$ from $W(S)$ corresponds to choosing element $s_i$ from the set $S$. We note a simple reduction from Multicoloured Independent Set to MinCSP($R^{\vee}_{\neq,\neq}, =$). Let the input be a graph $G$ with $k$ colour classes $V(G) = V_1 \cup \ldots V_k$. Create a choice gadget for every $V_i$ without soft $\neq$-constraints (but

keeping the crisp one), and set the budget to $2k$. For every pair $u \in V_i$, $v \in V_j$ for $i \neq j$ such that $uv \in E(G)$, add a crisp constraint $R^{\vee}_{\neq,\neq}(u, f(u), v, f(v))$. Due to the budget, $u = f(u)$ holds for at least one $u \in V_i$ for every $V_i$, corresponding to a selection, and the $R$-constraint prevents that $u = f(u)$ and $v = f(v)$, thereby blocking the combination of $u \in V_i$ and $v \in V_j$. We defer the details.

Having shown that MinCSP$(\Gamma, =, \neq)$ is W[1]-hard if $\Gamma$ is non-conjunctive, it remains to show hardness for languages which are conjunctive but not $\mathsf{NEQ}_3$ or split. We use the following W[1]-hard problem (see [22, Lemma 6.1]).

| Split Paired Cut | |
|---|---|
| Instance: | Graphs $G_1, G_2$, vertices $s_1, t_1 \in V(G_1)$, $s_2, t_2 \in V(G_2)$, a family of disjoint edge pairs $\mathcal{P} \subseteq E(G_1) \times E(G_2)$, and an integer $k$. |
| Parameter: | $k$. |
| Question: | Is there a subset $X \subseteq \mathcal{P}$ of size at most $k$ such that for both $i \in \{1, 2\}$, $\{e_i : \{e_1, e_2\} \in X\}$ is an $st$-cut in $G_i$? |

Assuming $\Gamma$ is conjunctive, associate with a relation $R(x_1, \ldots, x_r)$ a graph where for each pair $i, j \in [r]$ there is a blue edge $x_i x_j$ if $i \neq j$ and $R$ implies $x_i = x_j$, and a red edge $x_i x_j$ if $R$ implies $x_i \neq x_j$. Then the graph of a split relation $R(X, Y)$, $X = \{x_1, \ldots, x_p\}$ and $Y = \{y_1, \ldots, y_q\}$ is a blue clique on $X$ and a red biclique of $(X, Y)$, and the graph of $\mathsf{NEQ}_3$ is a red triangle. The remaining cases are precisely the cases when the graph contains two disjoint edges $x_1x_2$, $x_3x_4$ with no blue edges connecting their endpoints (such as $R^{\wedge}_{=,=}$, $R^{\wedge}_{=,\neq}$ or $R^{\wedge}_{\neq,\neq}$). There is a curious parallel to the characterization of FPT cases of Boolean MinCSP, in terms of 2CNF-definable relations whose *Gaifman graph* is $2K_2$-free [29]. We show hardness for all such cases. If both $x_1x_2$ and $x_3x_4$ are blue, a reduction is immediate from Split Paired Cut. We sketch the reduction for a more interesting case of MinCSP$(R^{\wedge}_{\neq,\neq}, =, \neq)$, and note that the reduction for MinCSP$(R^{\wedge}_{=,\neq}, =, \neq)$ is a variant of the above, and hence omitted in this version of the paper.

▶ **Lemma 8.** *MinCSP$(R^{\wedge}_{\neq,\neq}, =)$ is W[1]-hard.*

**Proof sketch.** Let $(G_1, G_2, s_1, t_1, s_2, t_2, \mathcal{P}, k)$ be an instance of Split Paired Cut. By the construction of [28, Lemma 5.7], assume $k = 2\ell$, and $\mathcal{F}_i$ for $i \in \{1, 2\}$ are $s_it_i$-maxflows in $G_i$ partitioning $E(G_i)$ into $k$ pairwise edge-disjoint paths. Construct an instance $(I, k')$ of MinCSP$(R, =, \neq)$ with $k' = 5\ell$ as follows. Start by creating a variable for every vertex in $V(G_1) \cup V(G_2)$ with the same name as the vertex. For each $i \in \{1, 2\}$, consider a path $P \in \mathcal{F}_i$, and let $p$ be the number of edges on $P$. Create a choice gadget $W(P)$ for every $P$ with variables $v^P_0, \ldots, v^P_p$ following the path, and fresh variables $v^P_j$ for $p < j \leq 2p$ added to the instance. Observe that variables may appear on several paths in $\mathcal{F}_i$. In particular, $v^P_0 = s_i$ and $v^P_p = t_i$ for every $P \in \mathcal{F}_i$, so we have crisp constraints $s_i \neq t_i$. Furthermore, since $\mathcal{F}_i$ partitions $E(G_i)$, the construction contains a copy of graphs $G_1$ and $G_2$ with equality constraints for edges. Now we pair up edges according to $\mathcal{P}$. For every pair $\{e_1, e_2\} \in \mathcal{P}$, let $P \in \mathcal{F}_1$ and $Q \in \mathcal{F}_2$ be the paths such that $e_1 \in P$ and $e_2 \in Q$, and suppose $e_1 = v^P_{i-1}v^P_i$ and $e_2 = v^Q_{j-1}v^Q_j$. Pair up soft constraints $v^P_i \neq f(v^P_i)$ and $v^Q_j \neq f(v^Q_j)$, i.e. replace individual constraints with one soft constraint $R(v^P_i, f(v^P), v^Q_j, f(v^Q_j))$. Finally, if an edge $uv \in E(G)$ does not appear in any pair of $\mathcal{P}$, make constraint $u = v$ crisp in $I$. This completes the construction.

The proof of correctness is deferred to the full paper. ◀

For all remaining cases, as noted, every $R \in \Gamma$ is either split or $\mathsf{NEQ}_3$, and there is a simple reduction from MinCSP($\Gamma$) to Triple Multicut. Since the latter is FPT (Theorem 10) the classification is complete.

We omit the details of the classification for singleton expansion cases. The proofs are somewhat more intricate, explicitly using the algebraic method as employed by Bodirsky, Chen and Pinsker [5], but ultimately both the hardness proofs and algorithmic cases are less interesting than for the above.

## 4 Triple Multicut

We show that Triple Multicut is in FPT, thus proving Theorem 10. The algorithm works by a reduction to Boolean MinCSP, i.e. MinCSP($\Delta$) for a constraint language $\Delta$ over the binary domain $\{0, 1\}$. Parameterized complexity of Boolean MinCSP was completely classified by [29]. As a result of our reduction, we obtain an instance where $\Delta$ is bijunctive, i.e. every relation in $\Delta$ can be defined by a Boolean formula in CNF with at most two literals in each clause. Define the *Gaifman graph* of a bijunctive relation $R$ with vertices $\{1, \ldots, r\}$, where $r$ is the arity of $R$, and edges $ij$ for every pair of indices such that $R(x_1, \ldots, x_r)$ implies a 2-clause involving $x_i$ and $x_j$. A graph is $2K_2$-free if no four vertices induce a subgraph with two independent edges.

▶ **Theorem 9** (Theorem 1.2 of [29]). *Let $\Delta$ be a finite bijunctive Boolean constraint language such that the Gaifman graphs of all relations in $\Delta$ are $2K_2$-free. Then MinCSP($\Delta$) is in FPT.*

We are ready to present the algorithm.

▶ **Theorem 10.** *Triple Multicut is fixed-parameter tractable.*

**Proof Sketch.** Let $(G, \mathcal{T}, k)$ be an instance of Triple Multicut. By iterative compression, we obtain $X_V \subseteq V(G)$ and $X_\mathcal{T} \subseteq \mathcal{T}$ such that $|X_V| + |X_\mathcal{T}| \leq k + 1$ and all components of $G - X_V$ intersect triples in $\mathcal{T} \setminus X_\mathcal{T}$ in at most one vertex. Moreover, by branching on the intersection, we can assume that a hypothetical optimal solution $(Z_V, Z_\mathcal{T})$ to $(G, \mathcal{T}, k)$ is disjoint from $(X_V, X_\mathcal{T})$. Let $X = X_V \cup \bigcup_{uvw \in X_\mathcal{T}} \{u, v, w\}$ and guess the partition of the vertices in $X$ into connected components of $G - Z_V$. Identify vertices that belong to the same component, and enumerate them via the bijective mapping $\alpha : X \to \{1, \ldots, d\}$. Observe that for every triple $uvw \in X_\mathcal{T}$, values $\alpha(u)$, $\alpha(v)$ and $\alpha(w)$ are distinct since $X_\mathcal{T} \cap Z_\mathcal{T} = \emptyset$. Create an instance $I_\alpha$ of Boolean MinCSP as follows.

1. Introduce variables $v_i$ and $\hat{v}_i$ for every $v \in V(G)$ and $i \in [d]$.
2. For every vertex $v \in V(G)$, add soft constraint $\bigwedge_{i<j}(\neg v_i \vee \neg v_j) \wedge \bigwedge_i (v_i \to \hat{v}_i)$.
3. For every vertex $v \in X$, add crisp constraints $v_{\alpha(v)}$, $\hat{v}_{\alpha(v)}$, and $\neg v_j$, $\neg \hat{v}_j$ for all $j \neq \alpha(v)$.
4. For every edge $uv \in E(G)$ and $i \in [d]$, add crisp constraints $\hat{u}_i \to v_i$ and $\hat{v}_i \to u_i$.
5. For every triple $uvw \in \mathcal{T}$ and $i \in [d]$, add soft constraints $(\neg \hat{u}_i \vee \neg \hat{v}_i) \wedge (\neg \hat{v}_i \vee \neg \hat{w}_i) \wedge (\neg \hat{u}_i \vee \neg \hat{w}_i)$.

This completes the reduction. We defer the correctness proof to the full version. ◀

## 5 Disjunctive Multicut

We show that Disjunctive Multicut is constant-factor fpt-approximable. Section 5 presents the main loop of the Disjunctive Multicut algorithm, while Section 5 is dedicated to the most technical subroutine of the algorithm that involves *randomized covering of shadow* [37].

## Main Loop of the Disjunctive Multicut Algorithm

Let $G$ be a graph with vertices $V(G) = V^{\infty}(G) \uplus V^1(G)$ partitioned into undeletable and deletable, respectively. A subset $L \subseteq \binom{V(G)}{2}$ of pairs is a *request list*, and a set of vertices $X \subseteq V(G)$ *satisfies* $L$ if there is a pair $st \in L$ separated by $X$. This includes the possibility that $s \in X$ or $t \in X$. For a graph $G$ and a collection of request lists $\mathcal{L}$, we let $\mathrm{cost}(G, \mathcal{L})$ be the minimum size of a set $X \subseteq V^1(G)$ that satisfies all lists in $\mathcal{L}$. Disjunctive Multicut asks, given an instance $(G, \mathcal{L})$, whether $\mathrm{cost}(G, \mathcal{L}) \leq k$.

Disjunctive Multicut problem generalizes not only Multicut (which is a special case with $d = 1$) but also $d$-Hitting Set. To see the latter, take an edgeless graph $G$ and make every request a *singleton*, i.e. a pair $ss$ for a vertex $s \in V(G)$. The only way to satisfy a singleton $ss$ is to delete the vertex $s$ itself, and the only way to satisfy a list of singletons is to delete one of the vertices in it.

The intuitive idea behind the approximation algorithm for Disjunctive Multicut is to iteratively simplify the instance $(G, \mathcal{L}, k)$, making it closer to Bounded Hitting Set after each iteration. Roughly, we make progress if the maximum number of non-singleton requests in a list decreases. In each iteration, the goal is to find a set of $O(k)$ vertices whose deletion, combined with some branching steps, simplifies every request list. This process can continue for $O(d)$ steps until we obtain an instance of Bounded Hitting Set, which can be solved in fpt time by branching. The instance may increase in the process, but finally we obtain a solution of cost $f(d) \cdot k$ for some function $f$. We do not optimize for $f$ in our proofs. Observe also that in the context of constant-factor fpt approximability, some dependence on $d$ is unavoidable since the problem with unbounded $d$ generalizes Hitting Set.

Formally, for a request list $L$, let $\mu_1(L)$ and $\mu_2(L)$ be the number of singleton and non-singleton cut requests in $L$, respectively. Define the measure for a list $L$ as $\mu(L) = \mu_1(L) + 3\mu_2(L) = |L| + 2\mu_2(L)$, and extend it to a collection of list requests $\mathcal{L}$ by taking the maximum, i.e. $\mu(\mathcal{L}) = \max_{L \in \mathcal{L}} \mu(L)$. Observe that $\mu(\mathcal{L}) \leq 3d$ for any instance of Disjunctive Multicut. Further, let $V(L) = \bigcup_{st \in L} \{s, t\}$ denote the set of vertices in a list $L$, and $\nu(\mathcal{L}) = \mu_1(L) + 2\mu_2(L)$ be an upper bound on the maximum number of variable occurrences in a list of $\mathcal{L}$. The workhorse of the approximation algorithm is the following lemma.

▶ **Lemma 11.** *There is a randomized algorithm* Simplify *that takes an instance $(G, \mathcal{L}, k)$ of Disjunctive Multicut as input, and in $O^*(2^{O(k)})$ time produces a graph $G'$ and a collection of requests $\mathcal{L}'$ such that $|V(G')| \leq |V(G)|$, $\nu(\mathcal{L}') \leq \nu(\mathcal{L})$, $|\mathcal{L}'| \leq k^2|\mathcal{L}|$, and $\mu(\mathcal{L}') \leq \mu(\mathcal{L}) - 1$. Moreover, the following holds.*

- *If $\mathrm{cost}(G, \mathcal{L}) \leq k$, then, with probability $2^{-O(k^2)}$, we have $\mathrm{cost}(G', \mathcal{L}') \leq 2k$.*
- *If $\mathrm{cost}(G, \mathcal{L}) > 3k$, then we have $\mathrm{cost}(G', \mathcal{L}') > 2k$.*

Randomization in Lemma 11 comes from the use of the *random covering of shadow* of [37, 18]. They also provide a derandomized version of this procedure, so our algorithm can be derandomized as well. We postpone the proof of Lemma 11 until Section 5 since it requires introduction of some technical machinery. For now, we show how to prove Theorem 12 using the result of the lemma.

▶ **Theorem 12.** *Disjunctive Multicut is fixed-parameter tractable.*

**Proof.** Let $(G, \mathcal{L}, k)$ be an instance of Disjunctive Multicut. Repeat the following steps until $\mu_2(\mathcal{L}) = 0$. Apply the algorithm of Lemma 11 to $(G, \mathcal{L}, k)$, obtaining a new graph $G$ and a new collection of lists $\mathcal{L}$, and let $(G, \mathcal{L}, k) := (G', \mathcal{L}', 2k)$. When $\mu_2(\mathcal{L}) = 0$, let

**Algorithm 1** Main Loop.

```
1: procedure SolveDJMC(G, L, k)
2:     while μ2(L) > 0 do
3:         (G, L) ← Simplify(G, L, k)
4:         if Simplify rejects then
5:             reject
6:         k ← 2k
7:     W ← {vv : v ∈ V(G)}
8:     if SolveHittingSet(W, L, k) accepts then
9:         accept
10:    else
11:        reject
```

$W = \{vv : v \in V(G)\}$ be the set of singleton cut requests for every vertex in $V(G)$. Check whether $(W, \mathcal{L}, k)$ is a yes-instance of Hitting Set – if yes, accept, otherwise reject. See Algorithm 1 for the pseudocode.

To argue correctness, let $(G, \mathcal{L}, k)$ be the input instance and $(G', \mathcal{L}', k')$ be the instance obtained after simplification. By induction and Lemma 11, we have $|V(G')| \leq |V(G)|$ and $\nu(\mathcal{L}') \leq \nu(\mathcal{L})$. Since $\nu(\mathcal{L}') \leq \nu(\mathcal{L}) \leq 2d$ and $\mu_2(\mathcal{L}) = 0$, every list in $\mathcal{L}$ has at most $2d$ requests. Let $r$ be the number of calls to Simplify performed by the algorithm. Note that $r \leq \mu(\mathcal{L}) \leq 3d$ since the measure decreases by at least one with each iteration, and define $k' = 2^r k$. The lists in $\mathcal{L}'$ only contain singletons, thus $(G', \mathcal{L}', k')$ is essentially an instance of Hitting Set with sets of size $2d$. Moreover, $|\mathcal{L}'| \leq k^{2r}|\mathcal{L}|$, so the number of lists is polynomial in $|\mathcal{L}|$. We can solve $(G', \mathcal{L}', k')$ in $O^*((2d)^{k'})$ time by branching (see, for example, Chapter 3 in [20]). For the other direction, suppose $\text{cost}(G, \mathcal{L}) \leq k$. By Lemma 11 and induction, we have $\text{cost}(G', \mathcal{L}') \leq 2^r k \leq k'$ with probability $2^{-O(rk^2)}$, and the algorithm accepts. If $\text{cost}(I) > 3k$, then $\text{cost}(G', \mathcal{L}') > k'$ and the algorithm rejects. ◀

## Simplification Procedure

In this section we prove Lemma 11. We start by iterative compression and guessing. Then we delete at most $k$ vertices from the graph and modify it, obtaining an instance amenable to the main technical tool of the section – the *shadow covering* technique.

### Initial Phase

Let $(G, \mathcal{L}, k)$ be an instance of Disjunctive Multicut. By iterative compression, assume we have a set $X \subseteq V(G)$ that satisfies all lists in $\mathcal{L}$ and $|X| = c \cdot k + 1$, where $c := c(d)$ is the approximation factor. Assume $Z$ is an optimal solution to $G$, i.e. $|Z| \leq k$ and $Z$ satisfies all lists in $\mathcal{L}$. Guess the intersection $W = X \cap Z$, and let $G' = G - W$, $X' = X \setminus W$, and $Z' = Z \setminus W$. Construct $\mathcal{L}'$ starting with $\mathcal{L}$ and removing all lists satisfied by $W$. Further, guess the partition $\mathcal{X} = (X_1, \ldots, X_\ell)$ of $X'$ into the connected components of $G' - Z'$, and identify the variables in each subset $X_i$ into a single vertex $x_i$, and redefine $X'$ accordingly. Note that the probability of our guesses being correct up to this point is $2^{-O(k \log k)}$. Also, these steps can be derandomized by creating $2^{O(k \log k)}$ branches.

Now compute a minimum $\mathcal{X}$-multiway cut in $G'$, i.e. a set $M \subseteq V^1(G')$ that separates every pair of vertices $x_i$ and $x_j$ in $X'$. Note that $Z'$ is a $\mathcal{X}$-multiway cut by the definition of $\mathcal{X}$, so $|M| \leq |Z'| \leq k$. Such a set $M$ can be computed in $O^*(2^k)$ time using the algorithm

of [21]. If no $\mathcal{X}$-multiway cut of size at most $k$ exists, then abort the branch and make another guess for $\mathcal{X}$. If an $\mathcal{X}$-multiway cut $M$ of size at most $k$ is obtained, remove the vertices in $M$ from $G$ and along with the lists in $\mathcal{L}'$ satisfied by $M$. This completes the initial phase of the algorithm. Properties of the resulting instance are summarized below.

▶ **Lemma 13** (Proof Omitted). *After the initial phase we obtain a graph $G'$, a family of list requests $\mathcal{L}'$, and subset of vertices $X' \subseteq V(G')$ such that $|V(G')| \leq |V(G)|$, $\nu(\mathcal{L}') \leq \nu(\mathcal{L})$, $\mu(\mathcal{L}') \leq \mu(\mathcal{L})$, and $|X'| \in O(k)$. The set $X'$ satisfies all lists in $\mathcal{L}'$ and intersects each connected component of $G'$ in at most one vertex. Moreover, the following hold.*

- $\mathrm{cost}(G, \mathcal{L}) \leq k + \mathrm{cost}(G', \mathcal{L}')$.
- *If* $\mathrm{cost}(G, \mathcal{L}) \leq k$, *then, with probability* $2^{-O(k \log k)}$, *we have* $\mathrm{cost}(G', \mathcal{L}') \leq k$. *Moreover, there is a set $Z' \subseteq V(G')$, $|Z'| \leq k$ that satisfies all lists in $\mathcal{L}'$ and is disjoint from $X'$.*

### Random Covering of Shadow

Random covering of shadow is a powerful tool introduced by [37] and sharpened by [18]. We use the latter work as our starting point. Although [18] present their theorems in terms of directed graphs, their results are applicable to our setting by considering undirected edges as bidirectional, i.e. replacing every edge $uv$ with a pair of antiparallel arcs $(u, v)$ and $(v, u)$. Consider a graph $G$ with vertices partitioned into deltable and undeletable subsets, i.e. $V(G) = V^1(G) \uplus V^\infty(G)$. Let $\mathcal{F} = (F_1, \ldots, F_q)$ be a family of connected subgraphs of $G$. An *$\mathcal{F}$-transversal* is a set of vertices $T$ that intersects every subgraph $F_i$ in $\mathcal{F}$. If $T$ is an $\mathcal{F}$-transversal, we say that $\mathcal{F}$ is *$T$-connected*. For every $W \subseteq V(G)$, the *shadow of $W$ (with respect to $T$)* is the subset of vertices disconnected from $T$ in $G - W$. We state it for the case $T \subseteq V^\infty(G)$ which suffices for our applications.

▶ **Theorem 14** (Random Covering of Shadow, Theorem 3.5 in [18]). *There is an algorithm* RandomCover *that takes a graph $G$, a subset $T \subseteq V^\infty(G)$ and an integer $k$ as input, and in $O^*(4^k)$ time outputs a set $S \subseteq V(G)$ such that the following holds. For any family $\mathcal{F}$ of $T$-connected subgraphs, if there is an $\mathcal{F}$-transversal of size at most $k$ in $V^1(G)$, then with probability $2^{-O(k^2)}$, there exists an $\mathcal{F}$-transversal $Y \subseteq V^1(G)$ of size at most $k$ such that*

1. *$Y \cap S = \emptyset$, and*
2. *$S$ covers the shadow of $Y$ with respect to $T$.*

The following consequence is convenient for our purposes.

▶ **Corollary 15** (Proof Omitted). *Let $S$ and $Y$ be the shadow-covering set and the $\mathcal{F}$-transversal from Theorem 14, respectively. Define $R = V(G) \setminus S$ to be the complement of $S$. Then $Y \subseteq R$ and, for every vertex $v \in R$, either $v \in Y$ or $v$ is connected to $T$ in $G - Y$.*

Note that if a vertex $v \in N(S)$ and $v$ is undeletable, then $v \in R$ and $v \notin Y$, hence $v$ is connected to $T$ in $G - Y$. Since $Y \cap S = \emptyset$, every vertex in $N(v) \cap S$ is also connected to $T$ in $G - Y$, so we can remove $N(v) \cap S$ from $S$ (and add it to $R$ instead). By applying this procedure to exhaustion, we may assume that no vertex in $N(S)$ is undeletable.

With the random covering of shadow at our disposal, we return to Disjunctive Multicut. By Lemma 13, we can start with an instance $(G, \mathcal{L}, k)$ and a set $X \subseteq V(G)$ such that $|X| \in O(k)$, $X$ satisfies all lists in $\mathcal{L}$, every connected component of $G$ intersects $X$ in at most one vertex, and there is an optimal solution $Z$ disjoint from $X$. Let $\mathcal{T} := \mathcal{T}(G, \mathcal{L}, X, Z)$ be the set of cut requests in $\bigcup \mathcal{L}$ satisfied by both $X$ and $Z$. Define $\mathcal{F}$ as the set of $st$-walks for all $st \in \mathcal{T}$. Observe that an $\mathcal{F}$-transversal is precisely a $\mathcal{T}$-multicut. Apply the algorithm from Theorem 14 to $(G, X, k)$. Since $X$ and $Z$ are $\mathcal{F}$-transversals and $|Z| \leq k$ by assumption,

Theorem 14 and Corollary 15 imply that we can obtain a set $R \subseteq V(G)$ in fpt time such that, with probability $2^{-O(k^2)}$, there is an $\mathcal{F}$-transversal $Y \subseteq R$ of size at most $k$, and every vertex in $R \setminus Y$ is connected to $X$ in $G - Y$.

For every vertex $v \in V^1(G) \setminus X$, define a set of vertices $R_v \subseteq R \setminus X$ as follows:

- if $v$ is disconnected from $X$, then let $R_v = \emptyset$;
- if $v \in N(X)$ or $v \in R$, then let $R_v = \{v\}$;
- otherwise, let $R_v = R \cap N(H)$, where $H$ is the component of $G[S]$ containing $v$.

Note that, by definition, the set $R_v$ is an $Xv$-separator in $G$. Moreover, we have ensured that $N(S)$ does not contain undeletable vertices, so $R_v$ does not contain undeletable vertices. In a certain sense, the sets $R_v$ are the only $Xv$-separators that $Y$ needs to use. This idea is made precise in the following lemma.

▶ **Lemma 16** (Proof Omitted). *Let $G$ be a graph. Let $X$ and $Y$ be disjoint subsets of $V(G)$ such that $X$ intersects every connected component of $G$ in at most one vertex. Suppose $R \subseteq V(G)$ is such that $Y \subseteq R$ and all vertices in $R \setminus Y$ are connected to $X$ in $G - Y$. If a vertex $s$ is disconnected from $X$ in $G - Y$, then $R_s \subseteq Y$.*

Now we compute a simplified collection of lists $\mathcal{L}'$. Start by adding all lists in $\mathcal{L}$ to $\mathcal{L}'$. Remove every singleton request $xx$ such that $x \in X$ from every list of $\mathcal{L}'$. For every list $L \in \mathcal{L}'$ not shortened this way, let $st \in L$ be a non-singleton cut request satisfied by $X$. Consider $R_s$ and $R_t$ and apply one of the following rules.

**(R1)** If $|R_s| > k$ and $|R_t| > k$, remove $st$ from $L$.
**(R2)** If $|R_s| \leq k$ and $|R_t| > k$, replace $L$ with sets $(L \setminus \{st\}) \cup \{aa\}$ for all $a \in R_s$.
**(R3)** If $|R_s| > k$ and $|R_t| \leq k$, replace $L$ with sets $(L \setminus \{st\}) \cup \{bb\}$ for all $b \in R_t$.
**(R4)** If $|R_s| \leq k$ and $|R_t| \leq k$, replace $L$ with sets $(L \setminus \{st\}) \cup \{aa, bb\}$ for all $a \in R_t$, $b \in R_t$.

Finally, make vertices in $X$ undeletable, obtaining a new graph $G'$. This completes the simplification step. Note that each list in $\mathcal{L}$ is processed once, so the running time of the last step is polynomial.

Now we prove some properties of $G'$ and $\mathcal{L}'$ obtained above. Note that $|V(G')| = |V(G)|$. Since every list in $\mathcal{L}$ is processed once and with at most $k^2$ new lists, the size of $|\mathcal{L}'|$ grows by a factor of at most $k^2$. To see that $\nu(\mathcal{L}') \leq \nu(\mathcal{L})$ and $\mu(\mathcal{L}') \leq \mu(\mathcal{L}) - 1$, observe that every reduction rule replaces a list $L$ with new lists with either one less non-singleton request (so $\mu_2$ decreases by at least 1), and adds up to two singleton requests (so $\mu_1$ increases by at most 2). Moreover, in every list of $\mathcal{L}$ there is a cut request satisfied by $X$, so no list of $\mathcal{L}$ remains unchanged in $\mathcal{L}'$. We state the remaining ingredients for proving correctness.

▶ **Lemma 17** (Proof Omitted). *If $\text{cost}(G, \mathcal{L}) \leq k$, then, with probability $2^{-O(k^2)}$, we have $\text{cost}(G', \mathcal{L}') \leq 2k$.*

▶ **Lemma 18** (Proof Omitted). *If $\text{cost}(G, \mathcal{L}) > 2k$, then we have $\text{cost}(G', \mathcal{L}') > 2k$.*

We are now ready to prove Lemma 11.

**Proof of Lemma 11.** Suppose $(G, \mathcal{L}, k)$ is a yes-instance of Disjunctive Multicut. By Lemma 13, after the initial phase we obtain $G'$, $\mathcal{L}'$ such that $|V(G')| \leq |V(G)|$, $\nu(\mathcal{L}') \leq \nu(\mathcal{L})$, $\mu(\mathcal{L}') \leq \mu(\mathcal{L})$, and $\text{cost}(G', \mathcal{L}') \leq k$. Moreover, we obtain a set $X' \subseteq V(G')$, $|X'| \in O(k)$, that satisfies all lists in $\mathcal{L}'$, intersects every component of $G'$ in at most one vertex, and is disjoint from an optimum solution $Z'$ to $(G', \mathcal{L}', k)$. Now we apply random covering of shadow and the list reduction rules to $G', \mathcal{L}', X'$, obtaining a new graph $G''$ and a new set of lists $\mathcal{L}''$. By Lemma 17, with probability $2^{O(-k^2)}$, we have $\text{cost}(G'', \mathcal{L}'') \leq 2k$. This proves one statement of Lemma 11. For the second statement of Lemma 11, suppose $\text{cost}(G'', \mathcal{L}'') \leq 2k$. By Lemma 18, we have $\text{cost}(G', \mathcal{L}') \leq 2k$. By Lemma 13, $\text{cost}(G'\mathcal{L}') \leq 2k$ implies that $\text{cost}(G, \mathcal{L}) \leq 2k + k \leq 3k$, and we are done. ◀

## References


1 Libor Barto, Andrei A. Krokhin, and Ross Willard. Polymorphisms, and how to use them. In *The Constraint Satisfaction Problem*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

2 Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Karthik C. S., Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *J. ACM*, 68(3):16:1–16:40, 2021.

3 Manuel Bodirsky. *Constraint satisfaction with infinite domains*. PhD thesis, Humboldt University of Berlin, Unter den Linden, Germany, 2004.

4 Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Lecture Notes in Logic (LNL). Cambridge University Press, 2021. doi:10.1017/9781107337534.

5 Manuel Bodirsky, Hubie Chen, and Michael Pinsker. The reducts of equality up to primitive positive interdefinability. *J. Symb. Log.*, 75(4):1249–1292, 2010.

6 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2008.

7 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 43(2):136–158, 2008.

8 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2):9:1–9:41, 2010.

9 Manuel Bodirsky and Marcello Mamino. Constraint satisfaction problems over numeric domains. In *The Constraint Satisfaction Problem*, volume 7 of *Dagstuhl Follow-Ups*, pages 79–111. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

10 Manuel Bodirsky, Barnaby Martin, and Antoine Mottet. Discrete temporal constraint satisfaction problems. *J. ACM*, 65(2):9:1–9:41, 2018.

11 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *J. ACM*, 62(3):19:1–19:52, 2015.

12 Édouard Bonnet, László Egri, and Dániel Marx. Fixed-parameter approximability of Boolean MinCSPs. In *24th Annual European Symposium on Algorithms (ESA 2016)*, pages 18:1–18:18, 2016.

13 Édouard Bonnet, László Egri, Bingkai Lin, and Dániel Marx. Fixed-parameter approximability of boolean MinCSPs, 2018. doi:10.48550/arXiv.1601.04935.

14 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. *SIAM J. Comput.*, 47(1):166–207, 2018. doi:10.1137/140961808.

15 Karl Bringmann, Danny Hermelin, Matthias Mnich, and Erik Jan Van Leeuwen. Parameterized complexity dichotomy for Steiner multicut. *Journal of Computer and System Sciences*, 82(6):1020–1043, 2016.

16 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *FOCS*, pages 319–330. IEEE Computer Society, 2017.

17 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

18 Rajesh Chitnis, Marek Cygan, Mohammataghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(4):1–28, 2015.

19 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification, 2020. arXiv:2001.06867.

20 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

**21** Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory (TOCT)*, 5(1):1–11, 2013.

**22** Konrad K Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. Almost consistent systems of linear equations. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3179–3217. SIAM, 2023.

**23** Eduard Eiben, Clément Rambaud, and Magnus Wahlström. On the parameterized complexity of symmetric directed multicut. In *IPEC*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

**24** Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020.

**25** Peter Jonsson. Constants and finite unary relations in qualitative constraint reasoning. *Artif. Intell.*, 257:1–23, 2018.

**26** Karthik C. S. and Subhash Khot. Almost polynomial factor inapproximability for parameterized k-clique. In *CCC*, volume 234 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

**27** Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.

**28** Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Solving hard cut problems via flow-augmentation. *arXiv e-prints*, pages arXiv–2007, 2020.

**29** Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation III: complexity dichotomy for boolean CSPs parameterized by the number of unsatisfied constraints. In *SODA*, pages 3218–3228. SIAM, 2023.

**30** Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. Flow-augmentation III: Complexity dichotomy for Boolean CSPs parameterized by the number of unsatisfied constraints. *arXiv preprint*, 2023. `arXiv:2207.07422`.

**31** Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. *SIAM J. Comput.*, 46(3):1087–1110, 2017.

**32** Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *Journal of the ACM (JACM)*, 67(3):1–50, 2020.

**33** Jason Li and Debmalya Panigrahi. Deterministic min-cut in poly-logarithmic max-flows. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 85–92. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00017`.

**34** Bingkai Lin, Xuandi Ren, Yican Sun, and Xiuhan Wang. Constant approximating parameterized $k$-SETCOVER is W[2]-hard. In *SODA*, pages 3305–3316. SIAM, 2023.

**35** Daniel Lokshtanov, Pranabendu Misra, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. FPT-approximation for FPT problems. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 199–218. SIAM, 2021.

**36** Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. `doi:10.1016/j.tcs.2005.10.007`.

**37** Daniel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *SIAM Journal on Computing*, 43(2):355, 2014.

**38** Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. *J. ACM*, 63(4):37:1–37:33, 2016.

**39** Magnus Wahlström. Quasipolynomial multicut-mimicking networks and kernels for multiway cut problems. *ACM Trans. Algorithms*, 18(2):15:1–15:19, 2022.

**40** Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020.

# Engineering Fast Algorithms for the Bottleneck Matching Problem

**Ioannis Panagiotas**
Neo4j, Malmö, Sweden

**Grégoire Pichon**
Université Claude Bernard Lyon 1 and LIP, France
UMR5668, CNRS, ENS de Lyon, Inria, UCBL1, France

**Somesh Singh**
CNRS and LIP, ENS de Lyon, France
UMR5668, CNRS, ENS de Lyon, Inria, UCBL1, France

**Bora Uçar**
CNRS and LIP, ENS de Lyon, France
UMR5668, CNRS, ENS de Lyon, Inria, UCBL1, France

## Abstract

We investigate the maximum bottleneck matching problem in bipartite graphs. Given a bipartite graph with nonnegative edge weights, the problem is to find a maximum cardinality matching in which the minimum weight of an edge is the maximum. To the best of our knowledge, there are two widely used solvers for this problem based on two different approaches. There exists a third known approach in the literature, which seems inferior to those two which is presumably why there is no implementation of it. We take this third approach, make theoretical observations to improve its behavior, and implement the improved method. Experiments with the existing two solvers show that their run time can be too high to be useful in many interesting cases. Furthermore, their performance is not predictable, and slight perturbations of the input graph lead to considerable changes in the run time. On the other hand, the proposed solver's performance is much more stable; it is almost always faster than or comparable to the two existing solvers, and its run time always remains low.

**2012 ACM Subject Classification** Theory of computation

**Keywords and phrases** bipartite graphs, assignment problem, matching

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.87

**Related Version** *Full Version*: `https://inria.hal.science/hal-04146298`

**Supplementary Material** *Software*: `https://gitlab.inria.fr/bora-ucar/bottled`
archived at `swh:1:dir:c711aeb1f701657080e12c716a5476361b8099ae`

## 1 Introduction

A matching in a graph is a set of edges without any common vertices. A maximum cardinality matching in a graph has the largest number of edges among all matchings. We investigate algorithms for finding a maximum cardinality matching whose minimum edge weight is the maximum on bipartite graphs with edge weights. This is called the bottleneck matching problem or linear bottleneck assignment problem when all vertices can be matched [3, Section 6.2]. Formally, the bottleneck matching problem is to find a maximum cardinality matching $\mathcal{M}$ which maximizes

$$\min_{(r_i,c_j)\in\mathcal{M}} w_{i,j}.$$

This problem can be solved in polynomial time.


The bottleneck matching problem arises in different contexts [3, Section 6.2.8]. We are motivated by the Birkhoff–von Neumann (BvN) decomposition of doubly stochastic matrices [2], which arises in practical applications [1, 4, 16, 21]. In this case, the bipartite graphs associated with matrices have equal number of vertices on both sides and contain perfect matchings. A known heuristic for the BvN decomposition [10] repeatedly calls a bottleneck matching algorithm on the bipartite graph of a dynamically changing matrix.

**Table 1** Run time, in seconds, of MC64J2 and MC64J3 on 12 problems. $\mathbf{A}$ is the original matrix, $\mathbf{AP}$ is the same matrix with a random column permutation; $\mathbf{S} = \mathbf{D}P(\mathbf{A})\mathbf{E}$, where $P(\mathbf{A})$ is the 0-1 matrix with 1s at the nonzero positions of $\mathbf{A}$; and $\mathbf{D}$ and $\mathbf{E}$ are positive diagonal matrices scaling $P(\mathbf{A})$ to be doubly stochastic.

| | **A** | | **AP** | | **S** | |
|---|---|---|---|---|---|---|
| matrix | J2 | J3 | J2 | J3 | J2 | J3 |
| atmosmodm | 0.05 | 0.26 | 0.13 | 0.40 | 684.87 | 6487.07 |
| CurlCurl_3 | 0.05 | 0.31 | 0.15 | 0.48 | 578.83 | 2958.07 |
| ss | 2.28 | 0.90 | 2317.46 | 1.73 | 810.10 | 16501.00 |
| vas_stokes_2M | 0.25 | 1.87 | 6310.44 | 727.57 | 1420.01 | 4571.14 |

The software MC64 [7, 8] is the state-of-the-art and implements two algorithms, denoted MC64J2 and MC64J3, for the bottleneck matching problem. To the best of our knowledge these are the only available codes that can handle bipartite graphs corresponding to large sparse matrices. Their worst-case run time are $O(n(m+n)\log_2 n)$ and $O(nm\log_2 n)$, on bipartite graphs with $n$ vertices on each side and $m$ edges [8]. MC64, especially the newer MC64J2 [8], is well engineered. It works very well for graphs corresponding to matrices from numerical applications. However, it does not have stable run time behavior in two senses. First, when run on the same bipartite graph twice with different edge weights the difference in run time can be in the order of hours. Second, on two equivalent problem instances, where one is obtained from the other by just reordering the vertices, the run time can change dramatically. We report the run time of MC64 on four matrices, from the SuiteSparse Matrix Collection [5], in Table 1 to explain this – more experiments of similar nature are in Section 4. Here, the set of rows and the set of columns of a matrix correspond to the two parts of the bipartite graph with an edge between two vertices if the corresponding entry in the matrix is nonzero, and the nonzero values are the edge weights. The table contains results for $\mathbf{A}$, for $\mathbf{AP}$ where $\mathbf{P}$ is a random permutation, and for the doubly stochastic matrix $\mathbf{S}$ which is obtained by scaling the pattern of $\mathbf{A}$ with Sinkhorn-Knopp algorithm [20].

The bipartite graphs of $\mathbf{A}$ and $\mathbf{AP}$ are the same apart from renumbering of the vertices in one part. While on $\mathbf{A}$ both MC64J2 and MC64J3 are fast, both methods suffer on $\mathbf{AP}$; the run time of MC64J2 is not acceptable for the last two instances, and that of MC64J3 is high for the last one in Table 1. The bottleneck matching problems on $\mathbf{A}$ and $\mathbf{AP}$ are essentially the same, as permuting the columns does not change the values, nor the bottleneck matching and its value. The bipartite graph of $\mathbf{S} = \mathbf{D}P(\mathbf{A})\mathbf{E}$ is the same as that of $\mathbf{A}$ with different edge weights, hence the problems are not equivalent. Now, the run time of both methods is too much for all instances. The wildly varying run time of both MC64 routines on $\mathbf{S}$ in comparison to those on $\mathbf{A}$ further highlight the instability in their performance.

Our aim in this paper is to develop an algorithm for the bottleneck matching problem which is better than the state-of-the-art codes in MC64. For this purpose, we study an overlooked alternative from the literature. We make observations that pave the way for an efficient algorithm, implement and compare it against the codes from MC64. We conduct a large set of experiments to show that our approach is usually much faster than MC64 and in addition exhibits stable and robust performance.

Section 2 gives a brief background and a summary of the known algorithms. Section 3 contains the proposed algorithm. Section 4 presents the experimental results, and Section 5 concludes the paper. Appendix of the related version, which is available at `https://inria.hal.science/hal-04146298`, contains detailed description of the experiments and further experiments.

## 2 Background and related work

A matrix $\mathbf{A}$ with $nnz(\mathbf{A})$ nonzero entries can be represented with a bipartite graph $G = (R \cup C, E)$ where each row of $\mathbf{A}$ corresponds to a unique vertex in $R$, each column of $\mathbf{A}$ corresponds to a unique vertex in $C$, and there is an edge $(r_i, c_j)$ whenever $a_{ij} \neq 0$. The number $m$ of edges in $G$ is thus equivalent to $nnz(\mathbf{A})$. When the edges are weighted, the weight of the edge $(r_i, c_j)$ is $|a_{ij}|$, that is the magnitude of the nonzeros of $\mathbf{A}$. For a vertex $v$, we use $\text{adj}(v)$ to denote the set of its neighbors.

A *matching* is a set of edges with no common vertices. A matching is of *maximum cardinality* if it has the largest number of edges. Given a matching $\mathcal{M}$, a vertex is *matched* if an edge from $\mathcal{M}$ is incident on it and *free* otherwise. A matching is *perfect* if it matches all vertices. The *deficiency* of a matching $\mathcal{M}$ is the difference between the maximum cardinality of a matching and $|\mathcal{M}|$. Given a matching $\mathcal{M}$ in the graph $G$, a path in $G$ is $\mathcal{M}$-*alternating* if its edges are alternately in $\mathcal{M}$. An $\mathcal{M}$-alternating path $\mathcal{P}$ is $\mathcal{M}$-*augmenting* if the start and end vertices of $\mathcal{P}$ are both free. A *vertex cover* is a set of vertices that includes at least one vertex from each edge. In a bipartite graph the maximum cardinality of a matching is equal to the minimum cardinality of a vertex cover [3, Th. 2.7].

Given a bipartite graph, any of its maximum cardinality matchings can be used to obtain a canonical decomposition called Dulmage-Mendelsohn (DM) decomposition [11]. Based on the DM decomposition, Pothen and Fan [18] describe algorithms to permute sparse matrices in a block upper triangular form (BTF):

$$\mathbf{A} = \begin{array}{c} \\ H_R \\ S_R \\ V_R \end{array} \begin{array}{c} \begin{array}{ccc} H_C & S_C & V_C \end{array} \\ \begin{pmatrix} \mathbf{A}_H & * & * \\ O & \mathbf{A}_S & * \\ O & O & \mathbf{A}_V \end{pmatrix} \end{array}. \tag{1}$$

In a BTF, the submatrix $\mathbf{A}_H$ has more columns than rows, and all rows in $H_R$ are matched to a column in $H_C$ in any maximum cardinality matching; the submatrix $\mathbf{A}_S$ is square with at least one perfect matching; the submatrix $\mathbf{A}_V$ has more rows than columns, and all columns in $V_C$ are matched to a row in $V_R$. The rows/columns in each block are defined as follows

$H_R = \{$row vertices reachable from free column vertices via alternating paths$\}$,
$H_C = \{$free column vertices or column vertices reachable from free column vertices via alternating paths$\}$,
$V_R = \{$free row vertices or row vertices reachable from free row vertices via alternating paths$\}$,
$V_C = \{$column vertices reachable from free row vertices via alternating paths$\}$,
$S_R = R \setminus (H_R \cup V_R)$, and
$S_C = C \setminus (H_C \cup V_C)$.

A standard BFS/DFS-based graph traversal algorithm will find these sets in linear time.

We make some observation on the BTF form of a matrix. First, the DM decomposition reveals a minimum cover [3, Alg. 3.1]. As all nonzeros in the BTF (1) are confined in the rows $H_R \cup S_R$ and the columns in $V_C$, the vertex set $\mathcal{C} = H_R \cup S_R \cup V_C$ is a cover. Since the cardinality of $\mathcal{C}$ is equal to the maximum cardinality of a matching, it is a minimum cover. Second, if one adds new nonzeros to the diagonal blocks, upper diagonal blocks, or to the blocks $(S_R, H_C)$ and $(V_R, S_C)$, the maximum cardinality of a matching does not change. This is so, as the new nonzeros cannot create augmenting paths.

Hall's theorem [13] states that for a bipartite graph $G$ to have a column-perfect matching, the relation $|S| \leq |\bigcup_{c \in S} \text{adj}(c)|$ must hold for any subset $S$ of columns. If a similar relation holds for all subsets of rows, then $G$ will have perfect matchings.

An $n \times n$ matrix $\mathbf{A} \neq 0$ is called *doubly stochastic* if every entry is nonnegative and the sum of entries in each row and each column is equal to 1. Any nonnegative square matrix, whose bipartite graph has perfect matchings, can be scaled with two diagonal matrices to be doubly stochastic [20]. A *permutation matrix* is a square matrix where each row/column contains exactly one nonzero value equal to 1. A perfect matching in the bipartite graph representation of $\mathbf{A}$ corresponds to a permutation matrix. The bipartite graphs of doubly stochastic matrices have perfect matchings.

Henceforth, we assume that the given bipartite graph contains perfect matchings. We comment on rectangular matrices and matrices without perfect matchings in Section 3.3.

## 2.1 Related work

We review three algorithms from the literature [3, 7, 8]. The first two are implemented in MC64, and to the best of our knowledge are currently the best practical algorithms. Burkard et al. [3, Section 6.2.4] describe two other algorithms [12, 19], which are more theoretical.

### 2.1.1 Shortest-augmenting path based algorithms

Algorithms based on shortest augmenting paths start with a matching which has the maximum bottleneck value for the currently matched vertices $C'$ in one part, say $C$. In order to augment the matching, a shortest augmenting path from a free vertex $c$ of $C$ is found with a variant of Dijkstra's shortest path algorithm. Augmenting along the shortest paths maintains the invariant that the current matching has the maximum bottleneck value for any matching that matches the vertices $C' \cup \{c\}$. The process continues until a perfect matching is obtained.

The state-of-the-art implementation in MC64 [8], MC64J2, starts by computing an upper bound $\omega$ on the bottleneck value, which is the minimum of maximum in each column and row. It then computes a maximal matching on the graph containing only edge weights no smaller than $\omega$, which is then improved by length-three augmenting paths in a preprocess step. Then, a shortest-augmenting path is sought from each free column vertex to solve the problem. MC64J2 implements an efficient adaptation of Dijkstra's algorithm to find these paths. Depending on the edge weights, the structure of the bipartite graph, or the visit order many edges and vertices may be visited while finding an augmenting path. As seen in Table 1, this can accumulate and result in very long run time.

### 2.1.2 Threshold-based algorithms

Let $G$ be a weighted bipartite graph. For a value $\omega$, let $G[\omega]$ contain only the edges of $G$ with weight at least $\omega$. Threshold-based algorithms find the largest $\omega$ for which $G[\omega]$ has a perfect matching. They do so by considering different values for $\omega$, testing whether $G[\omega]$

has a perfect matching or not and tuning the next $\omega$ to a higher or lower value accordingly. MC64's implementation, MC64J3, discusses initialization and algorithmic choices to reduce the number of tests [7]. However, MC64 uses a depth-first-search algorithm to find the augmenting paths for each test which can slow it down, as seen in Table 1.

### 2.1.3 An algorithm based on duality

This algorithm is also threshold-based and uses the duality of matchings and coverings to find the next threshold. We explain this algorithm in more detail as ours improves upon it. Let $\omega$ be a value where $G[\omega]$ does not contain a perfect matching. Let $\mathcal{M}$ be a maximum cardinality matching in $G[\omega]$. From $\mathcal{M}$ one can define a minimum vertex cover $\mathcal{C} = H_R \cup S_R \cup V_C$ of $G[\omega]$ with vertex sets $I = H_R \cup S_R$ and $J = V_C$ [3, Section 6.2.3]. Since $\mathcal{M}$ is not perfect, there must be an edge in $G$ from a vertex in $\bar{I} = R \setminus I$ to another vertex in $\bar{J} = C \setminus J$. The value $\max_{i \in \bar{I}, j \in \bar{J}} a_{ij}$ thus cannot be smaller than the bottleneck matching value of $G$ and can be used as the next threshold. This approach is shown in Algorithm 1 [3, Section 6.2.3].

**Algorithm 1** Duality-based algorithm.

**Input** : $G$, an edge weighted bipartite graph having perfect matchings
**Output** : $\mathcal{M}$, a bottleneck perfect matching

Let $\omega$ be an upper bound on the bottleneck matching value
$\mathcal{M} \leftarrow \emptyset$
**while** $|\mathcal{M}| < n$ **do**
  $\mathcal{M} \leftarrow$ a maximum cardinality matching in $G[\omega]$
  **if** $|\mathcal{M}| < n$ **then**
1  Let $I \subseteq R$ and $J \subseteq C$ be the vertex sets of the associated cover of $G[\omega]$
2  $\omega \leftarrow \max_{i \in R \setminus I, j \in C \setminus J} w_{i,j}$ `/* in G, not G[ω] */`

The maximum cardinality matching in $G[\omega]$ can be found in $O(\sqrt{n}\,\mathrm{nnz}(\mathbf{A}[\omega]))$ time in the worst case [14]. Once such a matching is found, the associated minimum cover and the maximum uncovered value at Line 2 can be obtained in linear time. Therefore, the worst case time complexity of Algorithm 1 is $O(\sqrt{n}\,\mathrm{nnz}(\mathbf{A}))$ times the number of iterations. The worst case run time for Algorithm 1 can hence be too high. This is so as a new edge, due to the reduced $\omega$, does not mean one more edge in the maximum matching, and hence the while loop can even run for more than $n$ iterations.

## 3 The proposed algorithm

Our algorithm is based on Algorithm 1 and integrates threshold techniques. Let $G = (R \cup C, E)$ be an edge weighted bipartite graph, with $w_{i,j}$ being the weight of the edge $(r_i, c_j)$. Let $\omega$ be a nonnegative value, $G[\omega]$ be as before, and the bottleneck matching value $b^\star$ be the largest $\omega$ for which $G[\omega]$ has a perfect matching. We call a value $\omega$ *safe*, when $\omega \geq b^\star$. Algorithm 1 produces decreasing safe values that converge to $b^\star$. We make a series of observations to find $b^\star$ faster. The first observation is that there are several minimum vertex covers associated with a given maximum cardinality matching. Using the BTF (1), let $\mathcal{C}' = H_R \cup S_C \cup V_C$. As all nonzeros are covered by $\mathcal{C}'$ and $|\mathcal{C}'| = |H_R \cup S_R \cup V_C|$, $\mathcal{C}'$ is also a minimum cover. If we choose this cover, the sets $I$ and $J$ at Line 1 of Algorithm 1 become $H_R$ and $S_C \cup V_C$, in which case, the maximum value can be different. This leads to the following proposition.

▶ **Proposition 1.** *Let $\mathcal{C}_1 = H_R \cup S_R \cup V_C$ and $\mathcal{C}_2 = H_R \cup S_C \cup V_C$ be two minimum covers of $G[\omega]$ revealed by the BTF (1). Let $\omega_1$ and $\omega_2$ be the maximum values defined in Line 2 of Algorithm 1 for $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. Then, $\min(\omega_1, \omega_2)$ is safe.*

**Proof.** Let $\omega$ be the current value, and $\omega_1 < \omega_2$ without loss of generality. This means that $\omega_2$ is in $\mathbf{A}(V_R, S_C)$ and all entries in $\mathbf{A}(V_R, H_C)$ are smaller than $\omega_2$. The maximum cardinality of a matching in $G[\omega_2]$ cannot be larger than that in $G[\omega]$, as the set $\mathcal{C}_1 = H_R \cup S_R \cup V_C$ still covers all edges of $G[\omega_2]$, including those that arise in $\mathbf{A}(S_R, H_C)$. Hence the cover $\mathcal{C}_1$ can be used to get the next $\omega$ in Line 2, which concludes the proof. ◀

Based on Proposition 1, one can use the smaller of the largest uncovered element in $(S_R \cup V_R, H_C)$ and that in $(V_R, H_C \cup S_C)$. We propose to exploit the two identified covers as much as possible for a faster convergence of $\omega$ to $b^\star$. As we reason in Lemma 2 below, one can find a tighter bound on $b^\star$, depending on the deficiency of the current matching as well as the (original) adjacencies of the vertices in the identified covers.

▶ **Lemma 2.** *Let $\mathcal{M}$ be a maximum cardinality matching in $G[\omega]$ with a deficiency of $k$ in $G$; $\mathbf{A}[\omega]$ and $\mathbf{A}$ be, respectively, the matrices associated with $G[\omega]$ and $G$; $\mathcal{C} = H_R \cup S_C \cup V_C$ be a minimum vertex cover of $G[\omega]$ associated with $\mathcal{M}$ when $\mathbf{A}[\omega]$ is permuted in a BTF (1); Let $\omega_k^r$ be the $k$th largest element in $\bigcup_{i \in S_R \cup V_R} \max\{w_{i,j} : j \in H_C\}$, and $\omega_k^c$ the $k$th largest element in $\bigcup_{j \in H_C} \max\{w_{i,j} : i \in S_R \cup V_R\}$. Then, $\omega_k = \min(\omega_k^r, \omega_k^c)$ is safe.*

**Proof.** Consider first the set $H_C$ of columns, and note that $|H_C| - |H_R| = k$ as all other columns are matched. By Hall's theorem, $|H_C| \leq |\bigcup_{c \in H_C} \text{adj}(c)|$ must hold in $\mathbf{A}$ as there is a perfect matching. Among all rows in $\bigcup_{c \in H_C} \text{adj}(c)$, we have $|H_R|$ in the set $H_R$. Therefore there must be at least $k$ other nonzero rows in $\mathbf{A}(S_R \cup V_R, H_C)$. The element $\omega_k^r$ from $\bigcup_{i \in S_R \cup V_R} \max\{w_{i,j} : j \in H_C\}$ is safe as any value greater than that will cover less than $k$ rows and Hall's conditions cannot be satisfied. A similar argument applies to $\omega_k^c$ by considering the set $S_R \cup V_R$ of rows. In $\mathbf{A}[\omega]$ we have $\text{adj}(S_R \cup V_R) = S_C \cup V_C$, and $|S_R \cup V_R| - |S_C \cup V_C| = k$. The element $\omega_k^c$ must be safe since we need at least $k$ nonzero columns in $\mathbf{A}(S_R \cup V_R, H_C)$. The value $\omega_k$ is thus safe as the minimum of two safe values. ◀

One can identify several minimum covers, collect the $k$th largest uncovered element with respect to each, and use the minimum of the collected elements as the next $\omega$. As finding these minimum covers can be expensive, we propose using the two which are readily revealed by the BTF. That is, we use $\omega = \min(\omega_1, \omega_2)$ where

$$\begin{aligned} \omega_1^r, \omega_1^c &= k\text{-th largest row and column maximum entries in } \mathbf{A}(S_R \cup V_R, H_C)\,, \\ \omega_1 &= \min(\omega_1^r, \omega_1^c)\,, \qquad (2) \\ \omega_2^r, \omega_2^c &= k\text{-th largest row and column maximum entries in } \mathbf{A}(V_R, H_C \cup S_C)\,, \\ \omega_2 &= \min(\omega_2^r, \omega_2^c)\,. \qquad (3) \end{aligned}$$

This corresponds to applying Hall's theorem to the sets $H_C$ and $H_C \cup S_C$ of columns and to the sets $V_R$ and $S_R \cup V_R$ of rows.

## 3.1 Putting it all together

The proposed algorithm BOTTLED is shown in Algorithm 2. The input is a sparse matrix represented in the compressed storage by columns (CSC) format. BOTTLED creates a compressed storage by rows (CSR) representation of the input matrix. It then sorts the nonzeros in each row and each column in non-increasing order of their values. Then the threshold $\omega$ is initialized as the minimum of the $2n$ nonzero values consisting of the maximum in each row and maximum in each column. The algorithm then updates the threshold $\omega$ in a while-loop as in Algorithm 1. In the while-loop there are three subroutines:

MaximumCardinalityMatching, SAP, and DM-dec. These subroutines correspond to a maximum cardinality matching algorithm, a shortest augmenting path-based method to match a column, and an algorithm obtaining the row and column blocks of the BTF (1).

**Algorithm 2** BOTTLED: The proposed bottleneck matching algorithm.

**Input** : $\mathbf{A}$, stored by columns
**Output**: $\mathcal{M}$, a bottleneck perfect matching

Create a CSR representation of $\mathbf{A}$
Sort the nonzeros in each row and in each column in non-increasing order of values
1 $\omega \leftarrow$ min of the maximum in each row, maximum in each column
$G[\omega] \leftarrow (R \cup C, \emptyset)$ and $\mathcal{M} \leftarrow \emptyset$
**while** $|\mathcal{M}| < n$ **do**
2 for each $i$, release new $a_{ij} \geq \omega$ and for each $j$ release new $a_{ij} \geq \omega$ into $G[\omega]$
**if** $|\mathcal{M}| = n - 1$ **then**
$\mathcal{M} \leftarrow \text{SAP}(G, \mathcal{M}, c)$ with the last free vertex $c$
**else**
3 $\mathcal{M}' \leftarrow \text{MaximumCardinalityMatching}(G[\omega], \mathcal{M})$
**if** $|\mathcal{M}'| = n$ **then** $\mathcal{M} \leftarrow \mathcal{M}'$; break
**if** $|\mathcal{M}'| = |\mathcal{M}|$ **then**
4 select a vertex $c$
$\mathcal{M} \leftarrow \text{SAP}(G, \mathcal{M}', c)$
**else**
$\mathcal{M} \leftarrow \mathcal{M}'$
$\langle H_R, S_R, V_R, H_C, S_C, V_C \rangle \leftarrow \text{DM-dec}(G[\omega], \mathcal{M})$
5 $\omega \leftarrow \min(\omega_1, \omega_2)$ with $\omega_1$ as in (2) and $\omega_2$ as in (3)

Algorithm 2 stores the edges of $G[\omega]$ over the storage of $\mathbf{A}$ in the CSC and CSR formats, without explicitly building adjacency lists. The start address of each row and column are the same as those of $\mathbf{A}$. For each row/column of $G[\omega]$, we keep an end-pointer which points to the smallest nonzero of $\mathbf{A}$ in that row/column that is no smaller than $\omega$. These end-pointers are initialized before the while-loop in $O(n)$ time, and incremented at Line 2 at each iteration of the while-loop. Therefore the total run time cost of building $G[\omega]$s is $O(\text{nnz}(\mathbf{A}))$.

In Algorithm 2, $\text{SAP}(G, \mathcal{M}, c)$ refers to the algorithm summarized in Section 2.1.1. As stated before, SAP needs the current matching to have the bottleneck value among all matchings covering the same set of column vertices. The approach outlined in Algorithm 1 produces such matchings, that is why, at any point, one can resort to SAP. We invoke SAP in two cases: (i) when the deficiency is one; (ii) when an update of $\omega$ did not yield an increase in the cardinality of the current matching. The first case is straightforward. For the second case, we apply a simple heuristic to help the algorithm converge faster. As any free column vertex can be the start of an augmenting path, we choose $c \in C$ whose largest edge weight not included in the current $G[\omega]$ is minimum. Matching $c$ will lead to a reduced $\omega$, and the reduction will hopefully be large with this choice of $c$ (some empirical results are in the appendix of the related version). The most common algorithms for the maximum cardinality matching problem take an initial matching as input and augment it. This is very suitable at Line 3 of Algorithm 2, as we have a maximum cardinality matching on a graph, we add new edges, and then ask for a maximum cardinality matching in the new graph. We have used the code-base of MatchMaker [6, 15] to implement this step in the implicit representation of $G[\omega]$.

At Line 5 of Algorithm 2, we use a binary heap with a limit $k$ on its size. For $\omega_1^r$, the nonzeros of each row in $S_R \cup V_R$ with value smaller than $\omega$ are visited, and the largest element is used as a key in the heap. When the heap is full, keys are added only if they are larger than the current minimum, which is then removed. The minimum of the heap is returned as

$\omega_1^r$. Other quantities of (2) and (3) are computed similarly. All nonzeros of $\mathbf{A} - \mathbf{A}[\omega]$ in the rows $S_R \cup V_R$ and columns $H_C$ can be visited and the heap operations can be performed in $O(\text{nnz}(\mathbf{A}(S_R \cup V_R, H_C)) + (|H_C| + |S_R \cup V_R|) \log k)$ time. A similar analysis holds for $\mathbf{A}(S_R \cup V_R, H_C)$. The run time of one iteration of the while loop is thus dominated by that of the maximum cardinality matching algorithm. We do not have an estimate on the number of iterations of the while-loop; in the empirical results in Section 4.3 the number is less than what a binary search approach would yield.

## 3.2 In the context of a BvN decomposition method

The Birkhoff–von Neumann (BvN) theorem [2] states that a doubly stochastic matrix $\mathbf{A}$ can be written as $\mathbf{A} = \sum_{i=1}^{\ell} \alpha_i \mathbf{P}_i$ where each $\mathbf{P}_i$ is a permutation matrix, and $\alpha_i$s are positive coefficients summing up to one. Such a decomposition is not unique, and the problem of finding a decomposition with the smallest $\ell$ value is NP-Complete [10].

One heuristic [10] for obtaining a BvN decomposition of a double stochastic matrix $\mathbf{A}$ works as follows. It finds the value $b$ of a bottleneck matching whose pattern is the permutation matrix $\mathbf{P}$, replaces $\mathbf{A}$ with $\mathbf{A} - b\mathbf{P}$, and continues until a zero matrix is obtained. A property of this heuristic is that the successive bottleneck values are in a non-increasing order [9]. Our bottleneck matching algorithm is very fitting in this case. One can create the CSR representation and sort each row and column once. Then, executing the while loop of Algorithm 2 will obtain a bottleneck matching for the current matrix. Once $b$ and $\mathbf{P}$ are obtained, replacing $\mathbf{A}$ with $\mathbf{A} - b\mathbf{P}$ can be done by subtracting $b$ from each matched entry and updating that entry's position in the sorted list of both rows and columns in overall $O(n + \text{nnz}(\mathbf{A}))$ time. While doing so, one can update the end-pointers used for $G[\omega]$, and avoid the preprocessing in Algorithm 2 at subsequent invocations.

## 3.3 Rectangular matrices or matrices without perfect matchings

The case in which there are perfect matchings in the given bipartite graph is common in applications where the bipartite graphs correspond to sparse matrices. This is especially so in the BvN decomposition, which was our motivation. Nonetheless MC64's J2 and J3 work for cases in which one part of the bipartite graph has more vertices than the other, where the smaller side can be perfectly matched. This corresponds to $n_R \times n$ matrices for $n_R > n$ that have column-perfect matchings. Our algorithm can handle this case either by initializing $\omega$ at Line 1 using only the column values, or by using those and only the $n$th maximum of the set of $n_R$ maximum entries, one from in each row.

Consider now the most general case corresponding to $n_R \times n$ matrices, with $n_R \geq n$ and without column perfect matchings. In this case, MC64J2 returns *a maximum cardinality matching*, without necessarily finding the correct bottleneck value. That is so because not all vertices from which the shortest-augmenting paths are sought can be matched in a bottleneck maximum cardinality matching. We do not know of a suitable fix for this. MC64J3 and its equivalent THRESH on the other hand work correctly. The presented algorithm BOTTLED needs four minor modifications to handle this general case: (i) the maximum size of a matching $n'$ should be computed at the beginning; (ii) the initialization should use the choose the smallest of the $n'$th maximum of $n$ maximum entries, one from each column, and the $n'$th maximum of the $n_R$ maximum entries, one from each row; (iii) at Line 5, the deficiency is $k = n' - |\mathcal{M}|$; and (iv) the shortest-augmenting path method should not be used.

# 4 Experiments

We observed in a preliminary set of experiments that MC64J2 is generally faster than MC64J3; the geometric mean of the ratios of the run time of the latter to that of the former was 4.6. As already highlighted in Table 1, the run time of MC64J3 can be too large to experiment (this happens more frequently than with MC64J2). That is why we have implemented a threshold-based approach, referred here as THRESH, using the same code base of BOTTLED, and use it in our experiments instead of MC64J3. THRESH uses the initialization procedure common to MC64J3 and BOTTLED to find a large initial matching and an upper bound $\omega$ on the bottleneck value. If this matching is not maximum, then the edge weights are sorted to find the bottleneck value by a binary search between the smallest edge weight and the initial value $\omega$. The binary search uses only the available edge weights and each time half of the edges are discarded. Our codes, written in C, are available at `https://gitlab.inria.fr/bora-ucar/bottled`; the codes used in the experiments are elsewhere [17].

The next subsection describes the data set, and Section 4.2 conducts a performance analysis of BOTTLED. Section 4.3 compares BOTTLED with MC64J2 and THRESH. Last, some experiments using BOTTLED within a BvN decomposition heuristic are discussed in Section 4.4. Appendix of the related version contains detailed information about experiments.

## 4.1 Data set and measurements

We have experimented with all square matrices with perfect matchings, at least 100,000 rows, less than 250,000,000 nonzeros, and with no explicit zeros from SuiteSparse [5]. There were 113 such matrices at the time of experimentation. From each matrix, we created six types of problem instances, which are denoted as $\mathbf{A}, \mathbf{DAE}, \mathbf{D}P(\mathbf{A})\mathbf{E}, \mathbf{AP}, \mathbf{DAPE}$, and $\mathbf{D}P(\mathbf{A})\mathbf{PE}$. The **A**-type instance corresponds to the bipartite graph of the original matrix with the magnitudes of the entries as edge weights. The **DAE**-type and $\mathbf{D}P(\mathbf{A})\mathbf{E}$-type instances correspond, respectively, to the scaled version of the matrices and their patterns with 20 iterations of the Sinkhorn–Knopp [20] algorithm, and the other three types of instances are obtained from the first three by random column permutations. We discarded the instances **A** and **AP** for 0-1 matrices; for the same set of matrices **DAE**-type and $\mathbf{D}P(\mathbf{A})\mathbf{E}$-type instances are identical, and hence we kept only one of them. We discarded the instances in which the initialization algorithm found the bottleneck value, in which case all three methods are equivalent. We report the experiments with 14 **A**-type, 18 **DAE**-type, 58 $\mathbf{D}P(\mathbf{A})\mathbf{E}$-type instances, and the same number of instances with column permutations. For each instance, each algorithm is run five times and the geometric mean of the run time is reported as that algorithm's performance on that instance; when column permutations are applied, these correspond to five different permutations. When comparing two algorithms' run time, we do not include cases where both algorithms run in less than one second (as both are very small and the difference between the algorithms is insignificant). Appendix of the related version contains all the results.

We carry out the experiments on a machine having Intel(R) Xeon(R) CPU E7-8890 v4 with a clock-speed of 2.20 GHz, and 1.5TB memory. The machine runs Debian GNU/Linux 11 (64 bit). All the codes are compiled with `GCC` version 10.2.1, with option `-O3`. We ran MatchMaker [6, 15] to verify that there were perfect matchings. We used MatchMaker with options "no cheap matching", "Push-Relabel + fairness" as the core algorithm. For the sake of completeness, we report that the maximum run time of MatchMaker for any matrix was 2.65 seconds for vas_stokes_4M and 28.19 for the column permuted version of the same matrix.

## 4.2 Performance analysis of BOTTLED

We first analyze the percentage of the total run time of BOTTLED spent in the preprocessing step – creating the CSR representation or sorting the entries. Table 2 summarizes the results for six different problem instances, where BOTTLED took more than one second for **A**-, **DAE**- and/or **D***P*(**A**)**E**-type instances. This table presents the geometric mean of the percentage of the time spent creating the CSR representation and sorting (with respect to the total time of BOTTLED). The row "tTime" contains the geometric mean of the run time of BOTTLED on the instances **A**, **DAE** and **D***P*(**A**)**E** in seconds, and for the instances **AP**, **DAPE** and **D***P*(**A**)**PE** it contains the geometric mean of the ratio of the run time of BOTTLED on the permuted instances to the original ones. For example, the run time under the column **D***P*(**A**)**PE** is obtained by taking the geometric mean of ratio of the run time of BOTTLED on the 33 **D***P*(**A**)**PE** instances to that on the 33 **D***P*(**A**)**E** instances.

As we can see from Table 2, the average run time of BOTTLED on any of the six problem instance types is below 10 seconds. A first observation is that for all instances, the two preprocessing steps account for a non-negligible part of the total run time. At two extremes, they take 7% and 37% of the total time in **D***P*(**A**)**E** and **DAPE** instances, respectively. We further observe that the absolute run time of CSR and sorting increases for the permuted instances. The percentage of the total time spent in CSR also increases for the permuted instances while that of sorting either increases or remains the same. If the CSR representation is available, its creation can be skipped and one can reduce the run time considerably. As discussed in Section 3.2, for the targeted BvN application BOTTLED can skip not only the CSR creation, but also the sorting phase across different runs of the algorithm.

**Table 2** Percentage of the total time spent in creating a CSR matrix and sorting the nonzeros in BOTTLED; "tTime": the geometric mean of the run time of BOTTLED on the instances **A**, **DAE** and **D***P*(**A**)**E** in seconds, and the ratios of the others to their counterparts.

| | **A** | **AP** | **DAE** | **DAPE** | **D***P*(**A**)**E** | **D***P*(**A**)**PE** |
|---|---|---|---|---|---|---|
| CSR | 5% | 13% | 7% | 25% | 4% | 17% |
| sort | 5% | 5% | 10% | 12% | 3% | 3% |
| tTime | 6.54(s) | 1.28 | 3.43(s) | 1.02 | 3.40(s) | 1.28 |

**Table 3** Breakdown of the total time (tTime) of BOTTLED in seconds.

| matrix | instance-type | tTime | CSR | sort |
|---|---|---|---|---|
| vas_stokes_4M | **A** | 14.81 | 6.70 | 3.17 |
| | **AP** | 32.36 | 20.13 | 5.51 |
| | **DAE** | 23.44 | 6.93 | 3.66 |
| | **DAPE** | 52.99 | 19.93 | 5.57 |
| vas_stokes_2M | **D***P*(**A**)**E** | 11.67 | 2.96 | 2.01 |
| | **D***P*(**A**)**PE** | 19.30 | 8.81 | 2.57 |

Another observation from Table 2 is that the run time of BOTTLED consistently increases for the permuted instances. To put this into perspective, we present the run time of BOTTLED on a few instances in Table 3. We see that the increase in run time for the permuted matrices can be attributed, in part, to the creation of the CSR. For example, for vas_stokes_4M, excluding CSR and sort times from the total time yields 4.94 and 6.72 seconds for the while loop for **A**- and **AP**-type respectively. A similar calculation for the pairs **D***P*(**A**)**E** and **D***P*(**A**)**PE** for vas_stokes_2M shows that the while loop takes 6.7 and 7.92 seconds, respectively. In the instance pairs **DAE** and **DAPE** for vas_stokes_4M, the increase in the CSR time is still a contributing factor. From these two tables, we conclude that BOTTLED's preprocessing takes up a significant portion of the total run time.

We have also investigated the number of iterations of the while-loop of BOTTLED in different types of instances to see how stable and robust it is with respect to random column permutations. The number of iterations of the while loop were almost always the same for all 90 instances; in a few cases there was a difference of one in the number of iterations. The small changes in the number of iterations confirm the robustness of BOTTLED.

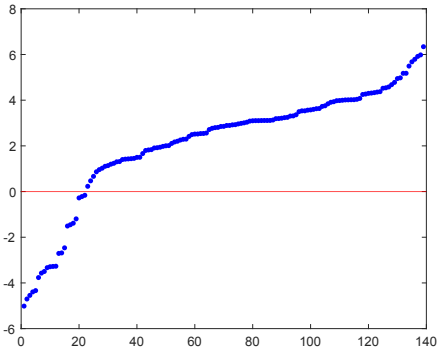
**Table 4** The geometric mean and the maximum of the run time of the three methods on six different instance types.

| | | **A** | **AP** | **DAE** | **DAPE** | **D$P$(A)E** | **D$P$(A)PE** |
|---|---|---|---|---|---|---|---|
| geomean | MC64J2 | 0.49 | 6.23 | 0.16 | 0.74 | 12.47 | 20.16 |
| | THRESH | 2.58 | 3.80 | 1.09 | 1.54 | 2.24 | 3.16 |
| | BOTTLED | 1.23 | 2.38 | 0.44 | 0.81 | 1.23 | 2.07 |
| maximum | MC64J2 | 123.04 | 18054.00 | 31.42 | 9139.63 | 1419.99 | 6813.20 |
| | THRESH | 43.55 | 64.90 | 59.42 | 91.74 | 63.51 | 77.57 |
| | BOTTLED | 14.52 | 31.81 | 23.22 | 51.55 | 58.64 | 76.20 |

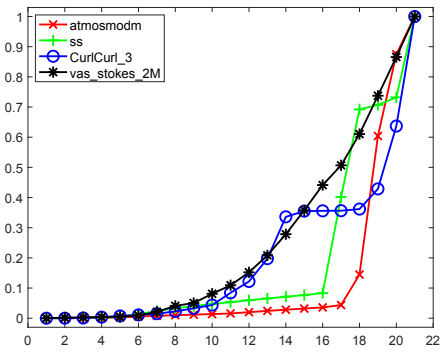
## 4.3 Comparison of different algorithms

We compared BOTTLED against MC64J2 and THRESH. We first provide a broad overview of the comparisons in Table 4, which lists the geometric mean of the run time of the three methods as well as their maximum run time in different problem instance types (all instances). As can be seen, BOTTLED exhibits the best performance overall, despite being slightly slower on average for **A**-, **DAE**-, **DAPE**- instance types than MC64J2– the margin is not large enough for MC64J2 to make up for the performance loss on the other instance types. Furthermore, BOTTLED's maximum run time is consistently smaller than those of both MC64J2 and THRESH. In contrast, the maximum run time of MC64J2 is prohibitive in all but the **A** and **DAE**-type instances. While THRESH avoids the prohibitive run time of its equivalent MC64J3 (see Table 1) by using a better cardinality matching algorithm [15], its maximum run time is still noticeably larger than that of BOTTLED in all but two cases.

We now look more into the performance of BOTTLED versus that of MC64J2. Figure 1a shows the natural logarithm (in the $y$-axis) of the ratio of the run time of MC64J2 to that of BOTTLED for the instances on which either method has a run time greater than 1 second. Here, MC64J2 is faster than BOTTLED on 22 instances, and BOTTLED is faster on the remaining 117 instances. In all 139 instances, the geometric mean of the ratio of the run time of MC64J2 to that of BOTTLED is 8.5, confirming that BOTTLED is faster than MC64J2 in average.




**(a)** The natural logarithm of the ratio of the run time of MC64J2 to that of BOTTLED in the $y$-axis in sorted order on 139 problem instances in the $x$-axis.




**(b)** The run time behavior of MC64J2 on four D$P$(A)E-type instances measured at 20 uniform steps ($x$-axis), normalized to the total time ($y$-axis).

**Figure 1** Performance comparison between MC64J2 and BOTTLED, and investigation on MC64J2.

In order to put these numbers into a perspective with the run time, we present Table 5. Table 5 lists six matrices in which the ratio of the run time of MC64J2 to that of BOTTLED was the smallest for certain instance types with or without permutation. It next lists six

**Table 5** The run time of MC64J2 and BOTTLED on selected instances, where their ratios were the lowest or the highest in different problem instance types or the permuted versions.

| matrix | type | MC64J2 | BOTTLED | type | MC64J2 | BOTTLED |
|---|---|---|---|---|---|---|
| c-73b | **A** | 0.09 | 13.55 | **AP** | 0.10 | 9.42 |
| c-73 | **A** | 0.19 | 14.52 | **AP** | 0.34 | 14.68 |
| dielFilterV3real | **A** | 0.33 | 11.72 | **AP** | 0.81 | 21.48 |
| boyd2 | **D**$P$(**A**)**E** | 0.03 | 3.31 | **D**$P$(**A**)**PE** | 0.07 | 1.84 |
| boyd2 | **DAE** | 0.03 | 2.43 | **DAPE** | 0.10 | 1.50 |
| dielFilterV3clx | **A** | 0.12 | 3.97 | **AP** | 0.49 | 7.19 |
| atmosmodd | **D**$P$(**A**)**E** | 555.76 | 7.01 | **D**$P$(**A**)**PE** | 1757.81 | 12.11 |
| vas_stokes_1M | **D**$P$(**A**)**E** | 452.36 | 4.20 | **D**$P$(**A**)**PE** | 2427.71 | 7.42 |
| vas_stokes_2M | **D**$P$(**A**)**E** | 1419.99 | 11.93 | **D**$P$(**A**)**PE** | 6813.20 | 18.37 |
| ss | **D**$P$(**A**)**E** | 824.91 | 4.66 | **D**$P$(**A**)**PE** | 2621.10 | 10.79 |
| CurlCurl_3 | **D**$P$(**A**)**E** | 610.16 | 1.54 | **D**$P$(**A**)**PE** | 211.56 | 3.80 |
| atmosmodj | **D**$P$(**A**)**E** | 565.60 | 7.27 | **D**$P$(**A**)**PE** | 1683.32 | 11.97 |
| vas_stokes_4M | **DAE** | 0.83 | 23.22 | **DAPE** | 9139.63 | 51.55 |
| ss | **A** | 2.25 | 2.65 | **AP** | 2532.09 | 8.65 |
| vas_stokes_4M | **A** | 0.53 | 14.23 | **AP** | 18054.00 | 31.81 |

matrices with different instance types in which the ratio of the run time of MC64J2 to that of BOTTLED was the largest for the original matrices and three for the permuted ones (three others were already in the list). Here we see that MC64J2 can have equivalently good performance on both the **A**- and **AP**-type instances (see the first block in Table 5). Still, there were some **AP** instances where MC64J2 ran prohibitively long (see the last two rows with ss and vas_stokes_4M). MC64J2 also struggled on several **D**$P$(**A**)**E** and **D**$P$(**A**)**PE** instances as seen in the second block. The prohibitively high run time of MC64J2 highlight the issue in the approaches based on the shortest augmenting paths: one may visit many edges and vertices to find augmenting paths at different stages during the execution. As such a behavior is also seen for DFS-based-cardinality matching algorithms [6], it cannot be attributed on the particular implementation of the shortest path algorithm in MC64J2.

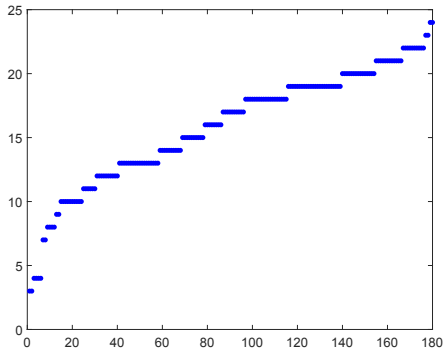
Figure 1b shows the run time behavior of MC64J2 on the four **D**$P$(**A**)**E**-type instances of Table 1. The total number of augmentations is divided by 20 to obtain a step size, the run time is measured after each step and normalized by the total time. As this figure shows, in some instances the augmentations took about the same time all throughout, whereas in others later augmentations took more time. As the greedy matching approach of MC64J2 does not always find a maximum matching, it can needlessly result in many additional augmentations. Those augmentations may lead to large run time, not solely because of their number. For example, in the **D**$P$(**A**)**E**-type instance of CurlCurl_3, a square matrix with $n = 1219574$ rows and nnz $= 13544618$ nonzeros, there are 20040 augmentations with a total run time of about 600 seconds, even though the maximum cardinality matching on $G[\omega_0]$ has a deficiency of one. Obviously, detecting this would lead to much better run time. On the other hand, for the **D**$P$(**A**)**E**-type instance of rajat31, a square martrix with $n = 4690002$ rows and nnz $= 20316253$ nonzeros, MC64J2 needs 1562500 augmentations (its greedy approach finds a maximum cardinality matching initially), and the whole run time is 48.39 seconds. As rajat31 is much bigger than CurlCurl_3 and needs more augmentations, its shorter run time attests that the process of augmenting one-by-one can take large time due to instance specific properties. It hence cannot constitute a reliable method for the bottleneck matching problem in general. In passing we note that MC64J2 is well-engineered and is faster than using our own SAP implementation for the augmentations.

We compare now BOTTLED and THRESH. In total, BOTTLED beat THRESH in 165 instances out of 180. The largest differences (in seconds) in favor of THRESH were in the **A**- and **AP**-type instances of c-73b and c-73. THRESH obtained a run time of 2.88 and 3.48 seconds

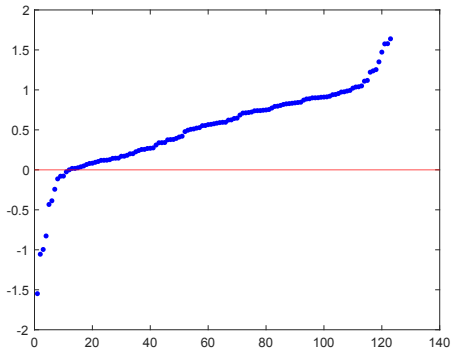
**Table 6** Run time, in seconds, of BOTTLED and the BvN decomposition heuristic, along with the number of permutation matrices in the decomposition.

| matrix | instance | BOTTLED initialization | longest run time of BOTTLED in an iteration | BvN num.perm | BvN time |
|---|---|---|---|---|---|
| atmosmodm | **DAE** | 0.22 | 58.73 | 46 | 452.14 |
| | **D**$P$(**A**)**E** | 0.21 | 14.43 | 50 | 200.80 |
| CurlCurl_3 | **DAE** | 0.35 | 17.30 | 50 | 183.43 |
| | **D**$P$(**A**)**E** | 0.34 | 4.33 | 50 | 68.45 |
| ss | **DAE** | 1.15 | 13.37 | 50 | 228.92 |
| | **D**$P$(**A**)**E** | 1.46 | 39.77 | 50 | 186.31 |
| vas_stokes_2M | **DAE** | 5.04 | 4.74 | 50 | 78.32 |
| | **D**$P$(**A**)**E** | 5.10 | 4.87 | 50 | 74.36 |

on c-73b, and 6.35 and 5.11 seconds on c-73. On these instances BOTTLED was relatively close to THRESH (see Table 5) – the largest difference is 10.67 seconds on the **A**-type instance of c-73b. Since both methods utilize the same core matching algorithm, the superiority of BOTTLED over THRESH should come from doing fewer iterations. Figure 2a supports this reasoning by plotting the difference between the number of iterations of THRESH and that of BOTTLED in nondecreasing order. As seen in this figure, THRESH's number of iterations is always larger than BOTTLED's; the theoretical observations of Section 3 translate to practical gains. We compute the ratio of the run time of THRESH to that of BOTTLED for the instances on which either THRESH or BOTTLED has a run time greater than 1 second (123 instances), and present the natural logarithm of this ratio in the $y$-axis in Figure 2b. As seen in this figure, BOTTLED fares better than THRESH in the majority of cases.




**(a)** The number of iterations of THRESH minus that of BOTTLED in the $y$-axis in sorted order on all 180 instances.




**(b)** The natural logarithm of the ratio of the run time of THRESH to that of BOTTLED in the $y$-axis in sorted order on 123 instances in the $x$-axis.

**Figure 2** Performance comparison between THRESH and BOTTLED.

## 4.4 Inside a BvN decomposition method

Table 6 presents the run time of the BvN decomposition method on the **DAE**- and **D**$P$(**A**)**E**-type instances of four matrices (obtained with 2500 scaling iterations). We run the BvN decomposition method until 50 permutation matrices or a coefficient of 0.92 are obtained. As each permutation matrix is obtained by a call to BOTTLED, we show their number in the column "num.perm". Further, the table also presents the time for the initial preprocessing of BOTTLED, and the maximum time taken in a call to BOTTLED subsequently for obtaining a

permutation matrix. We observe that the maximum run time of BOTTLED, in an iteration, multiplied by the number of calls is at least 2.92 (for **DAE** of ss) and up to 10.67 (for $\mathbf{D}P(\mathbf{A})\mathbf{E}$ of ss) times the total BvN time. This suggests that the run time of the subsequent bottleneck matching calls reduces appreciably, and BOTTLED works well inside the decomposition method by avoiding the preprocessing.

## 5 Conclusion

We have investigated the problem of finding a maximum bottleneck matching in bipartite graphs. Existing implementations for the problem suffer from unpredictable run time that can get prohibitively large, i.e., requiring thousands or even tens of thousands of seconds to complete. We have proposed a new algorithm called BOTTLED that converts an inefficient, duality-based approach into an efficient one through theoretical findings. Experimental results show that BOTTLED is almost always faster than the state-of-the-art methods. Furthermore, its run time is reliable and always remains within reasonable time limits. We have also explored its use inside a heuristic for the Birkhoff–von Neumann decomposition of doubly stochastic matrices and experimentally established the suitability of the proposed algorithm for this purpose.

Currently the proposed approach resorts to an augmenting-path-based method in few corner cases and only when there are perfect matchings. We plan to explore the possibility to use them more effectively, along with potential data reduction rules.

## References


1 M. Benzi and B. Uçar. Preconditioning techniques based on the Birkhoff–von Neumann decomposition. *Computational Methods in Applied Mathematics*, 17:201–215, 2017.

2 G. Birkhoff. Tres observaciones sobre el algebra lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–150, 1946.

3 R. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, Philadelphia, PA, USA, 2009.

4 C. Chang, W. Chen, and H. Huang. On service guarantees for input-buffered crossbar switches: A capacity decomposition approach by Birkhoff and von Neumann. In *Quality of Service, 1999. IWQoS '99. 1999 Seventh International Workshop on*, pages 79–86, 1999.

5 T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

6 I. S. Duff, K. Kaya, and B. Uçar. Design, implementation, and analysis of maximum transversal algorithms. *ACM Transactions on Mathematical Software*, 38:13:1–13:31, 2011.

7 I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(4):889–901, 1999.

8 I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22:973–996, 2001. doi:10.1137/S0895479899358443.

9 F. Dufossé, K. Kaya, I. Panagiotas, and B. Uçar. Further notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 554:68–78, 2018. doi:10.1016/j.laa.2018.05.017.

10 F. Dufossé and B. Uçar. Notes on Birkhoff–von Neumann decomposition of doubly stochastic matrices. *Linear Algebra and its Applications*, 497:108–115, 2016. doi:10.1016/j.laa.2016.02.023.

11 A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

**12** H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *J. Algorithms*, 9(3):411–417, 1988. Hopcroft-Karp algorithm can be used as an approximation algorithm p415.

**13** P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(37):26–30, 1935. URL: `http://jlms.oxfordjournals.org/cgi/reprint/s1-10/37/26.pdf`.

**14** J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

**15** K. Kaya, J. Langguth, F. Manne, and B. Uçar. Push-relabel based algorithms for the maximum transversal problem. *Computers & Operations Research*, 40(5):1266–1275, 2013.

**16** J. Kulkarni, E. Lee, and M. Singh. Minimum Birkhoff-von Neumann decomposition. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 343–354. Springer, 2017.

**17** I. Panagiotas, G. Pichon, S. Singh, and B. Uçar. Bottled: Fast algorithms for the bottleneck matching problem, April 2023. `doi:10.5281/zenodo.7871464`.

**18** A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16(4):303–324, December 1990. `doi:10.1145/98267.98287`.

**19** A. P. Punnen and K. Nair. Improved complexity bound for the maximum cardinality bottleneck bipartite matching problem. *Discret. Appl. Math.*, 55(1):91–93, 1994.

**20** R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific J. Math.*, 21:343–348, 1967.

**21** V. Valls, G. Iosifidis, and L. Tassiulas. Birkhoff's decomposition revisited: Sparse scheduling for high-speed circuit switches. *IEEE/ACM Transactions on Networking*, 29(6):2399–2412, 2021. `doi:10.1109/TNET.2021.3088327`.

# Subcubic Algorithm for (Unweighted) Unrooted Tree Edit Distance

**Krzysztof Pióro** ✉
Jagiellonian University, Kraków, Poland

## Abstract

The tree edit distance problem is a natural generalization of the classic string edit distance problem. Given two ordered, edge-labeled trees $T_1$ and $T_2$, the edit distance between $T_1$ and $T_2$ is defined as the minimum total cost of operations that transform $T_1$ into $T_2$. In one operation, we can contract an edge, split a vertex into two or change the label of an edge.

For the weighted version of the problem, where the cost of each operation depends on the type of the operation and the label on the edge involved, $\mathcal{O}(n^3)$ time algorithms are known for both rooted and unrooted trees. The existence of a truly subcubic $\mathcal{O}(n^{3-\epsilon})$ time algorithm is unlikely, as it would imply a truly subcubic algorithm for the APSP problem. However, recently Mao (FOCS'21) showed that if we assume that each operation has a unit cost, then the tree edit distance between two rooted trees can be computed in truly subcubic time.

In this paper, we show how to adapt Mao's algorithm to make it work for unrooted trees and we show an $\widetilde{\mathcal{O}}(n^{(7\omega+15)/(2\omega+6)}) \leq \mathcal{O}(n^{2.9417})$ time algorithm for the unweighted tree edit distance between two unrooted trees, where $\omega \leq 2.373$ is the matrix multiplication exponent. It is the first known subcubic algorithm for unrooted trees.

The main idea behind our algorithm is the fact that to compute the tree edit distance between two unrooted trees, it is enough to compute the tree edit distance between an arbitrary rooting of the first tree and every rooting of the second tree.

**2012 ACM Subject Classification** Theory of computation → Algorithm design techniques

**Keywords and phrases** tree edit distance, dynamic programming, matrix multiplication

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.88

**Related Version** *Full Version*: `https://arxiv.org/abs/2304.08632` [16]

**Acknowledgements** I would like to thank Adam Polak for introducing me to the topic, numerous discussions regarding the problem and his great help in editing this paper.

## 1 Introduction

The tree edit distance problem is a natural generalization of the classic string edit distance problem. Given two trees $T_1$ and $T_2$, the edit distance between $T_1$ and $T_2$ is defined as the minimum total cost of operations that transform $T_1$ into $T_2$. The exact definition of the problem depends on whether we consider rooted or unrooted trees.

For the unrooted variant, which is the focus of this paper, the trees are edge-labeled and for each vertex its neighbors form a cyclic order. We can think of unrooted trees as if they were embedded in the plane. In one operation, we can contract an edge, split a vertex into two or change the label of any edge. The cost of each operation depends on the type of the operation and the label on the edge involved.

Computing the edit distance between two trees has found applications in many different areas, such as computational biology [18, 11], image processing [1, 12, 14, 17] and comparing XML data [3, 4, 10]. One of the most notable examples is comparing the secondary structures of RNA molecules, which can be represented as rooted trees [18].


The tree edit distance problem has been already studied for many years. In 1977, Tai [19] showed the first algorithm that computes the edit distance between two rooted trees of size $n$ in $\mathcal{O}(n^6)$ time. Next, Zhang and Shasha [21] used a dynamic programming approach to improve the complexity to $\mathcal{O}(n^4)$ time. Their dynamic programming scheme was used as a basis for later algorithms. Klein [13] showed how to optimize their algorithm to $\mathcal{O}(n^3 \log n)$ time by better choosing the direction of transitions in their dynamic programming scheme. Furthermore, he showed that his algorithm can be extended to also work for the unrooted trees in the same time complexity. Later, Demaine, Mozes, Rossman and Weimann [7] further improved the time complexity for the rooted case to $\mathcal{O}(n^3)$. Finally, Dudek and Gawrychowski [8] showed that the tree edit distance between unrooted trees can also be solved in $\mathcal{O}(n^3)$ time.

From the lower bound side, Bringmann, Gawrychowski, Mozes and Weimann [2] showed that the existence of a truly subcubic $\mathcal{O}(n^{3-\epsilon})$ time algorithm for the tree edit distance is unlikely. They proved that the existence of such an algorithm implies a truly subcubic algorithm for the All Pairs Shortest Paths problem (assuming an alphabet of size $\Theta(n)$) and $\mathcal{O}(n^{k(1-\epsilon)})$ time algorithm for finding a maximum weight $k$-clique (assuming a sufficiently large constant size alphabet).

However, all of these previous algorithms and lower bounds work when the costs of the operations are arbitrary. Recently, Mao [15] showed that if we assume that each operation has unit cost, then the tree edit distance between two rooted trees can be computed in $\mathcal{O}(n^{2.9546})$ time. Since in the weighted setting it was possible to obtain the same time complexity for unrooted trees as for rooted trees, Mao posed the following open problem:

*Is it possible to compute the unweighted tree edit distance between two unrooted trees in subcubic time?*

We answer this question affirmatively.

## 1.1 Our contribution

We build on ideas of Mao [15] and Klein [13] and obtain the first ever known subcubic algorithm for the tree edit distance between unrooted trees in the unweighted setting. Our main result is the following:

▶ **Theorem 1.** *There is an $\widetilde{\mathcal{O}}(mn^{(5\omega+9)/(2\omega+6)}) \leq \mathcal{O}(mn^{1.9417})$ time algorithm that computes the (unweighted) tree edit distance between two unrooted trees of sizes $n$ and $m$.*

We were not able to adapt one of the optimizations from Mao, thus our algorithm is slightly slower than the one for rooted trees. Note that, however the numerical value of the exponent in our result is actually better than in the original Mao's paper. It is only due to the fact that we replaced the algorithm for max-plus multiplication of two bounded-difference $n \times n$ matrices that Mao used (running in $\mathcal{O}(n^{2.8244})$ time) with a more recent result from Chi, Duan, Xie and Zhang ($\mathcal{O}(n^{2.687})$ time) [6]. Dürr [9] showed that with this new result, Mao's algorithm works in $\widetilde{\mathcal{O}}(mn^{1.915})$ time. Thus, our algorithm is in fact slower than Mao's algorithm.

## 1.2 Technical overview

**Sketch of Mao's algorithm for rooted trees.** First, let us note that in the unrooted tree edit distance, we consider edge-labeled trees, but Mao's algorithm works for node-labeled trees. However, the modification of Mao's algorithm to work for rooted edge-labeled trees is

simple. Indeed, for both trees we can introduce a virtual root that becomes a parent of the original root and then we can associate the label of every vertex with the label of edge to its parent.

Mao's algorithm is based on Chen's algorithm [5]. Chen showed a completely different approach than previous algorithms that were based on Zhang and Shasha's dynamic programming scheme. He reduced the problem to the $(\min, +)$ matrix multiplication and obtained an algorithm working in $\mathcal{O}(n^4)$ time[1].

Mao builds on Chen's approach but considers an equivalent maximization problem of the similarity of trees. Due to this, the matrices occurring in his algorithm satisfy additional properties: they are monotone and the difference between two adjacent cells is bounded by a constant. For such matrices, there exists a truly subcubic time algorithm that computes the $(\min, +)$ product [6], which is crucial for obtaining a subcubic complexity in the tree edit distance problem.

In summary, Mao's algorithm, given two trees $T_1$ and $T_2$, computes matrices $S(T_1')$ for some subtrees $T_1'$ of tree $T_1$. Every such matrix $S(T_1')$ encodes the similarity between tree $T_1'$ and all relevant fragments of tree $T_2$. These fragments are defined as segments of the Euler tour sequence[2] and are formally defined in Definition 8.

Mao first shows a dynamic programming scheme based on Chen's algorithm that computes $S(T_1)$ in $\mathcal{O}(n^4)$ time. Next, he optimizes it to $\mathcal{O}(n^3)$ time by exploiting the special properties of similarity matrices. Then, to achieve a subcubic complexity, he presents a special decomposition scheme, which allows him to skip some of the subproblems. For block size $\Delta = n^d$ for some $d$ slightly smaller than 0.5, he decomposes the computation of $S(T)$ into $\mathcal{O}(n/\Delta)$ transitions of one of two types.

The first type is a concatenation of two trees. For that, he shows an $\mathcal{O}(t^{1-\epsilon}n^2)$ time algorithm that computes the $(\max, +)$ product of two bounded-difference matrices such that the entries in one of them are bounded by $t$. The second transition type involves expanding a subtree by adding a path going up from its root along with some additional subtrees attached to this path. For these transitions, Mao presents a special three-part combinatorial method.

**Sketch of our algorithm for unrooted trees.** To generalize Mao's algorithm to unrooted trees, we use the idea that Klein [13] used in his $\mathcal{O}(n^3 \log n)$ time algorithm. Klein used the fact that to compute the tree edit distance between two unrooted trees, it is enough to consider arbitrary rooting of the first tree and try all possible rootings of the second tree.

Direct application of this fact requires solving $\mathcal{O}(n)$ particular instances of the rooted tree edit distance. However, Klein showed that his $\mathcal{O}(n^3 \log n)$ time algorithm for the rooted tree edit distance can be modified to solve all of these instances at once in the same time complexity. To achieve this, he used the fact that his algorithm for rooted trees computes the edit distance between some fragments of the first tree and all subtrees of the second tree that correspond to some segment of the Euler tour sequence of that tree. Since the different rootings of the second tree correspond to different cyclic shifts of the Euler tour sequence, Klein modified his algorithm so that it computes the edit distance between some fragments of the first tree and all cyclic segments of the Euler tour sequence of the second tree.

We notice that Mao's algorithm has properties similar to those of Klein's algorithm. For some of the subtrees $T_1'$ of the first tree, it computes a similarity matrix $S(T_1')$ that encodes the similarity between tree $T_1'$ and all subtrees of $T_2$ that correspond to some segment of the

[1] By reduction we mean that he used $(\min, +)$ matrix multiplication as a sub-procedure in his algorithm.
[2] Actually, Mao used the so-called bi-order traversal sequence, but Euler tour sequence is its equivalent for edge-labeled trees.

Euler tour sequence of that tree. Thus, we need to modify the algorithm to handle cyclic segments of the Euler tour sequence. To do this, we build our new similarity matrices on a doubled Euler tour sequence (see Definitions 7–9 in Section 2). Now every cyclic segment of the original Euler tour occurs as a normal segment in our sequence.

The introduction of a doubled Euler tour requires us to make a few modifications to Mao's algorithm. The key technical challenge is handling of multiplication of new similarity matrices which we describe in Section 4.1.

## 2 Preliminaries

In the tree edit distance problem we consider ordered trees with labels on edges. We consider both rooted and unrooted trees. For rooted trees, ordered means that for each vertex we are given a left-to-right order of its children. For unrooted trees, ordered means that for each vertex its neighbors form a cyclic order.

Note that we can transform an unrooted tree into a rooted tree by choosing a root and the first edge to its children. This choice carries over to the other vertices and defines the order of their children. It uniquely determines the rooting of the tree, thus there are $2(n-1)$ possible rootings for an unrooted tree with $n$ vertices.

For simplicity, we first assume that both input trees are of equal size $n$, but at the end we address the case when they have different sizes.

As we are interested in unrooted tree edit distance, we consider only trees that are edge labeled (for rooted tree edit distance it is more common to have labels on nodes). In this paper, we assume that every edge is labeled from some alphabet $\Sigma$ of size $\mathcal{O}(n)$.

▶ **Definition 2** ((Unweighted) Tree Edit Distance). *Let $T_1$ and $T_2$ be rooted, ordered trees with labels on edges. We consider two types of operations:*

- *Label change of a selected edge in tree $T_1$ or $T_2$.*
- *Contraction of a selected edge in tree $T_1$ or $T_2$. When contracting an edge $pu$ where $p$ is parent of $u$, children of $u$ become children of $p$, they replace $u$ in the children's list of $p$ and keep their order.*

*The tree edit distance between $T_1$ and $T_2$, denoted by $\mathrm{ed}(T_1, T_2)$, is the minimum number of operations we have to perform on $T_1$ and $T_2$ to transform both trees into an identical tree.*

▶ **Definition 3** ((Unweighted) Unrooted Tree Edit Distance). *Let $T_1$ and $T_2$ be unrooted, ordered trees with labels on edges. We define the tree edit distance between $T_1$ and $T_2$ as the minimum edit distance over all possible rootings of $T_1$ and $T_2$.*

Klein [13] mentioned, it is enough to consider arbitrary rooting of the first tree and try all possible rootings of the second tree to find an optimal solution for the unrooted tree edit distance.

▶ **Lemma 4.** *Let $T_1$ and $T_2$ be unrooted trees. For every rooting of tree $T_1$ there is at least one rooting of $T_2$ that admits minimum edit distance between $T_1$ and $T_2$.*

Same as Mao, we consider a maximization problem equivalent to the tree edit distance problem.

▶ **Definition 5** (Similarity). *The similarity between two rooted trees $T_1$ and $T_2$ is defined as* $\mathrm{sim}(T_1, T_2) = |E(T_1)| + |E(T_2)| - \mathrm{ed}(T_1, T_2)$.

Similarity can also be interpreted as the weight of the heaviest matching between the edges of the tree $T_1$ and $T_2$, where the cost of edge matching is 2 when edges have the same labels and 1 when they are different. In addition, the matching must respect the tree structure, which means:

- if edge $a \in T_1$ is matched to edge $b \in T_2$, then edges in the subtree of $a$ can be matched only to edges in the subtree of $b$,
- if edge $a \in T_1$ is matched to edge $b \in T_2$ and $c \in T_1$ is matched to $d \in T_2$, then $a$ is "to the left" of $c$ if and only if $b$ is "to the left" of $d$.

Note that $0 \leq \mathrm{sim}(T_1, T_2) \leq 2\min(|E(T_1)|, |E(T_2)|)$.

▶ **Definition 6.** *By $\eta(e, f)$ we denote the cost of matching edges $e$ and $f$, that means $\eta(e, f) = 2$ if labels of these edges are equal and $\eta(e, f) = 1$ otherwise.*

**Tree definitions.** For a rooted tree $T$, by $L_T$ we denote the subtree of the first (leftmost) child of the root of $T$ along with the edge to the root of $T$. Similarly, by $R_T$ we denote the subtree of the last (rightmost) child of the root of $T$ along with the edge to the root of $T$. Given an edge $e$ in a rooted tree, we use $sub(e)$ to represent the subtree rooted at edge $e$..

For two rooted trees $T_1$ and $T_2$, by $T_1 + T_2$ we denote the tree formed by merging the roots of tree $T_1$ and $T_2$ such that edges from the tree $T_1$ are "to the left" of the edges from the tree $T_2$.

For two trees (rooted/unrooted) $T_1$ and $T_2$ such that $T_1 \subseteq T_2$, by $T_2 - T_1$ we denote the tree formed from tree $T_2$ by contracting all edges that appear in tree $T_1$.

To describe the "fragments" of a tree that we will consider in our algorithm, we use segments of the Euler cycle of the tree. Due to technical reasons, instead of dealing with a cyclic sequence, we consider a doubled Euler tour sequence.

▶ **Definition 7.** *Let $T$ be an unrooted tree. Consider the walk on this tree that starts at an arbitrary edge and goes twice through the Euler Tour, which visits neighbors according to their order.*

**(a)** *For $i \in \{1, \ldots, 4|E(T)|\}$ by $T(i)$ we denote the $i$-th edge of this walk.*
**(b)** *By $I(e)_i$ we denote the index of the $i$-th occurrence of the edge $e$ in this walk.*
**(c)** *By $l_{i,j}(e)$ we denote the index of the first occurrence of the edge $e$ in $T(i), \ldots, T(j-1)$.*
**(d)** *By $r_{i,j}(e)$ we denote the index of the second occurrence of the edge $e$ in $T(i), \ldots, T(j-1)$.*

Note that each edge appears exactly 4 times in this walk.

▶ **Definition 8** (Segment). *Given an unrooted tree $T$ and integers $l$, $r$ such that $1 \leq l \leq r \leq 4|E(T)|$ and $r - l \leq 2|E(T)|$, by $T[l, r)$ we denote the tree formed from the tree $T$ by contracting every edge that occurs less than 2 times in $T(l), \ldots, T(r-1)$.*

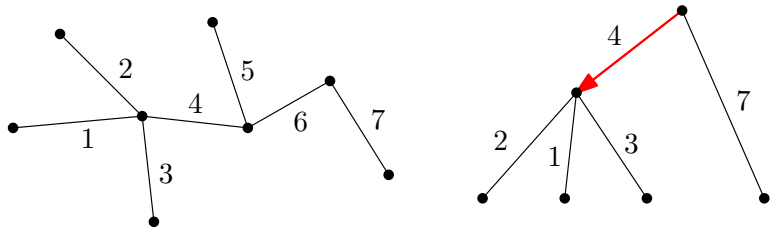
Let us note here that despite the fact that our input tree is unrooted, we can view the segments of this tree as rooted trees. Indeed, the first non-contracted edge that appears in the segment defines the rooting of this segment. Thus, the segments of length $2|E(T)|$ define all possible rootings of the tree $T$. See Figure 1.

**Matrix definitions.** Now we are ready to define similarity matrix – it encodes information about similarity of the whole rooted tree $T$ and all "relevant fragments" of unrooted tree $Q$.

▶ **Definition 9** (Similarity matrix). *Given a rooted tree $T$ and an unrooted tree $Q$ similarity matrix $S(T, Q)$ is a matrix of size $4|E(Q)| \times 4|E(Q)|$, where:*

$$S(T,Q)_{i,j} = \begin{cases} \mathrm{sim}(T, Q[i,j)) & \text{if } i \leq j \text{ and } j - i \leq 2|E(Q)| \\ -\infty & \text{if } i > j \text{ or } j - i > 2|E(Q)| \end{cases}$$

For simplicity, we also introduce additional notation for naming of similarity matrix cells. We divide them into three types:




1 3 3 4 6 7 7 6 5 5 4 2 2 1 1 3 3 4 6 7 7 6 5 5 4 2 2 1

**Figure 1** Example unrooted tree $T$ with its doubled Euler tour sequence and segment $T[11, 22)$ of this tree. The edge marked as red defines the rooting of this segment.

- *valid cells* – cells $S(T, Q)_{i,j}$ for which $i \leq j$ and $j - i \leq 2|E(Q)|$,
- *invalid cells* – cells $S(T, Q)_{i,j}$ for which $j - i > 2|E(Q)|$,
- *cells under diagonal*.

During our algorithm we will allow invalid cells to store values other than $-\infty$. Because of that, we introduce a special equality relation for similarity matrices:

▶ **Definition 10.** *By $=_{valid}$ we denote the relation on matrices which is true if and only if they are equal on the set of valid cells.*

▶ **Definition 11.** *Let $A$ and $B$ be $n \times n$ matrices. By $A \star B$ we denote their* $(\max, +)$ *product i.e.* $(A \star B)_{i,j} = max_{1 \leq k \leq n} A_{i,k} + B_{k,j}$.

Now, we define a few properties of matrices. Given an $n \times n$ matrix $A$ we call it:

**(a)** *finite-upper-triangular* matrix if all entries below diagonal are $-\infty$ and the rest are finite,
**(b)** *row-monotone* matrix if $A_{i,j} \leq A_{i,j+1}$ for all $i, j$,
**(c)** *column-monotone* matrix if $A_{i+1,j} \leq A_{i,j}$ for all $i, j$,
**(d)** *$W$-bounded-difference* matrix if for all $i, j$ we have:

$$|A_{i,j} - A_{i-1,j}| \leq W$$
$$|A_{i,j} - A_{i,j+1}| \leq W$$

**(e)** *bounded-difference* matrix if it is a $W$-bounded-difference matrix for some $W = \mathcal{O}(1)$,
**(f)** *finite-upper-triangular-$W$-bounded-difference* matrix if it is a finite-upper-triangular matrix and property d holds for all $i \leq j$,
**(g)** *$M$-bounded similarity matrix* if it is a similarity matrix and all finite entries are integers between 0 and $M$.

It is easy to see that for similarity matrices $S(T, Q)$ the properties row-monotone and column monotone hold for all cells except invalid cells. The following lemma, which is an analogy to Lemma 4.1 from Mao [15], tells that the finite-upper-triangular-2-bounded-difference property also holds for all cells except invalid cells.

▶ **Lemma 12.** *Given a rooted tree $T$ and an unrooted tree $Q$, we have:*

- $\text{sim}(T, Q[i, j + 1)) \leq \text{sim}(T, Q[i, j)) + 2$ *for all* $i \leq j, j - i + 1 \leq 2|E(Q)|$.
- $\text{sim}(T, Q[i - 1, j)) \leq \text{sim}(T, Q[i, j)) + 2$ *for all* $i \leq j, j - i + 1 \leq 2|E(Q)|$.

# 3 Cubic algorithm for Unrooted Tree Edit Distance

In this section, we introduce a basic dynamic programming scheme that computes the similarity of trees. It is the same scheme that Mao used, except that we use slightly different similarity matrices. We show how to efficiently compute this dynamic programming scheme in $\widetilde{\mathcal{O}}(n^3)$ time. This gives us a basic algorithm that will be later used in our subcubic algorithm.

As mentioned in Lemma 4, it is enough to consider an arbitrary rooting of the first tree and try all possible rootings of the second tree to find an optimal solution. Because of that, we can assume that we are given a rooted tree $T$, an unrooted tree $Q$ and we want to compute the edit distances between $T$ and every rooting of $Q$. To achieve that, we will compute the similarity matrix $S(T, Q)$. We will do this by computing recursively similarity matrices $S(T', Q)$ where $T'$ is some fragment of tree $T$. As the second argument of $S(T', Q)$ will always be $Q$ in our algorithm, we will use a shorthand $S(T') := S(T', Q)$. Additionally, we call $S(T')$ the similarity matrix of tree $T'$.

## 3.1 Dynamic programming scheme

To compute $S(T')$, we consider the following cases:

**(a)** $T'$ has no edges (it is a single vertex). Then the values in all valid cells are 0.

**(b)** The root of $T'$ has only one child. Let $r$ be the root and $u$ be its only child. We choose if we want to match $ru$ to some edge $e$ of $Q[i, j)$ or not:

$$S(T')_{i,j} = \max \begin{cases} S(T' - ru)_{i,j} \\ \max_{e \in Q[i;j)} \left\{ S(T' - ru)_{l_{i,j}(e)+1, r_{i,j}(e)} + \eta(ru, e) \right\} \end{cases}$$

**(c)** The root of $T'$ has more than one child. Let $R_{T'}$ be the subtree of the last child of the root of $T'$ along with the edge to the root of $T'$. Then:

$$S(T')_{i,j} = \max_{i \le k \le j} \{ S(T' - R_{T'})_{i,k} + S(R_{T'})_{k,j} \}$$

In other words: $S(T') = S(T' - R_{T'}) \star S(R_{T'})$.

Using this scheme, we can compute $S(T)$ in $O(n^4)$ time. Proof of the correctness of this algorithm is included in the full version of this paper [16].

## 3.2 Optimization to cubic

To optimize our algorithm to $\widetilde{\mathcal{O}}(n^3)$ time, we will exploit special properties of similarity matrices. We will rely on the fact that $S(T')$ is a $2|E(T')|$-bounded similarity matrix.

Furthermore, we will use the fact that these matrices are almost row-monotone and column-monotone. The only exception to this property are invalid cells, however their value is always $-\infty$. Because of that, we are able to use the same computation model that Mao used to store row-monotone, column-monotone matrices with slight modifications to handle invalid cells.

Let $l := 2n - 2$. We consider the following operations for $2l \times 2l$ matrices:

- create a new $2l \times 2l$ matrix $[-\infty]_{2l,2l}$,
- given a matrix $A$, create a copy of this matrix,

- given a matrix $A$, indexes $i', j'$ and a value $x$, create a new $2l \times 2l$ matrix $B := \text{rangemax}(A, i', j', x)$, such that:

$$B_{ij} = \begin{cases} \max(A_{ij}, x) & \text{if } i \leq i' \text{ and } j \geq j' \text{ and } j - i \leq l \\ A_{ij} & \text{otherwise} \end{cases}$$

Furthermore, given matrix $A$ we consider the following queries:

- get the value $A_{i,j}$,
- get the index $\text{mincol}(A, i, x) = \min\{j \mid A_{ij} \geq x\}$ or any index in $[1, 2l]$ if such an index does not exist,
- get the index $\text{maxrow}(A, j, x) = \max\{i \mid A_{ij} \geq x\}$ or any index in $[1, 2l]$ if such an index does not exist.

Note that the underlying data structure can keep a row-monotone, column-monotone matrix, and we can just modify queries to return $-\infty$ when reading invalid cells. This means we don't need to have the $j - i \leq l$ condition in the rangemax operation, thus we can use a simple 2D max operation (the same as Mao). All these operations can be performed with well-known data structures, such as persistent 2D segment trees in $\widetilde{\mathcal{O}}(1)$ time.

From now on, we assume that we store similarity matrices using the above computation model. We will go through all three cases of our dynamic programming scheme and we will show how to efficiently solve them using the above data structure:

**(a)** We can easily initialize all valid cells to 0 using $\mathcal{O}(n)$ rangemax queries. This gives us $\widetilde{\mathcal{O}}(n^2)$ time for the entire algorithm.

**(b)** Let us recall the equation for this case:

$$S(T')_{i,j} = \max \begin{cases} S(T' - ru)_{i,j} \\ \max_{e \in Q[i;j)} \left\{ S(T' - ru)_{l_{i,j}(e)+1, r_{i,j}(e)} + \eta(ru, e) \right\} \end{cases}$$

Naive computation of this equation relies on iterating through all cells we want to compute and for each $S(T')_{i,j}$ we iterate through each edge of $Q[i, j)$. Instead, we can do the opposite and iterate through each edge of $Q$ and update for each of them all relevant cells of $S(T')$. Thus, we first initialize $S(T')$ with $S(T' - ru)$ and then for each edge $e$ of $Q$ we make three rangemax operations. See Algorithm 1 for exact values of the parameters.

**Algorithm 1** Computation of case b.

**Input** $T, Q, S(T' - ru)$
**Output** $S(T')$

$S(T') \leftarrow S(T' - ru)$
**for all** $e \in E(Q)$ **do**
  $S(T') \leftarrow \text{rangemax}(S(T'), I(e)_1, I(e)_2 + 1, S(T' - ru)_{I(e)_1+1, I(e)_2} + \eta(ru, e))$
  $S(T') \leftarrow \text{rangemax}(S(T'), I(e)_2, I(e)_3 + 1, S(T' - ru)_{I(e)_2+1, I(e)_3} + \eta(ru, e))$
  $S(T') \leftarrow \text{rangemax}(S(T'), I(e)_3, I(e)_4 + 1, S(T' - ru)_{I(e)_3+1, I(e)_4} + \eta(ru, e))$
**return** $S(T')$

A single computation of this case takes $\widetilde{\mathcal{O}}(n)$ time, which gives us $\widetilde{\mathcal{O}}(n^2)$ time for the entire algorithm.

**(c)** This case is a matrix multiplication $S(T') = S(T' - R_{T'}) \star S(R_{T'})$. To speed up the computation, we will relay on the fact that $S(T')$ is a $2|E(T')|$ -bounded similarity matrix. The following lemma, which corresponds to Lemma 4.3 from Mao [15], shows that we can multiply similarity matrices in complexity dependent on the bounds of values of these matrices:

▶ **Lemma 13.** *Let $A, B$ be $l \times l$ matrices. If $A$ is a $t_A$-bounded similarity matrix and $B$ is a $t_B$-bounded similarity matrix, then we can compute $C = A \star B$ in $\widetilde{\mathcal{O}}(t_A t_B l)$ time.*

The algorithm that computes $C = A \star B$ is the same as Algorithm 1 from Mao [15] and can be found in the full version of this paper [16] together with the proof of correctness. Thus, every time we multiply similarity matrices of trees $T' - R_{T'}$ and $R_{T'}$, we contribute $\widetilde{\mathcal{O}}(|T' - R_{T'}||R_{T'}|n)$ time to the total time complexity of this case. To compute the sum of these components, we can correlate $|T' - R_{T'}||R_{T'}|$ with the number of pairs of edges $(e, f)$ where $e \in T' - R_{T'}$ and $f \in R_{T'}$. Let us sum these pairs over all multiplications. It is easy to see that each pair of edges from tree $T'$ appears at most once in such a sum. Thus, the total time complexity of this case is equal to $\widetilde{\mathcal{O}}(n^3)$.

After considering all three cases, we can see that the whole algorithm works in $\widetilde{\mathcal{O}}(n^3)$ time. Note here that if the input trees are of different sizes $n$ and $m$, then this algorithm works in $\widetilde{\mathcal{O}}(nm^2)$ time.

## 4 Subcubic algorithm for unrooted tree edit distance

In this section, we show how to get a subcubic algorithm for the unrooted tree edit distance problem. Our algorithm is based on the same decomposition scheme as Mao's algorithm. However, we will start by showing a different bottom-up view at $\widetilde{\mathcal{O}}(n^3)$ algorithm. This will give us a better intuition about Mao's decomposition scheme.

We can view the $\widetilde{\mathcal{O}}(n^3)$ algorithm as a decomposition scheme that computes $S(T)$ using the following two types of transitions:

- **Type I:** for trees $T_1, T_2$:
  compute $S(T_1 + T_2)$ from $S(T_1)$ and $S(T_2)$
- **Type II:** for trees $T_1, T_2$ such that $T_2$ is $T_1$ with one added edge going from the root up:
  compute $S(T_2)$ from $S(T_1)$

Note that the total time complexity of type I transitions is $\widetilde{\mathcal{O}}(n^3)$, while the total complexity of type II transitions is $\widetilde{\mathcal{O}}(n^2)$. This gives us an idea that to get a subcubic complexity we can try to balance these transitions. To reduce the number of type I transitions, we can generalize transitions of type II. Instead of adding a single edge going up from the root of $T_1$, we can add a "hat", that is, a path going up from the root along with some additional subtrees. This is the general idea of Mao's decomposition scheme, which we will now formally describe. See Figure 2 for an illustration of the "hat" structure.

First, we introduce additional definition that will help us define sub-problems occuring in the decomposition scheme. This definition corresponds to the synchronous subtree definition from Mao.

▶ **Definition 14** (Connected segment). *Given an unrooted tree $T$, segment $T[i, j)$ is called a connected segment if there is no edge that occurs in $T(i), \ldots, T(j-1)$ exactly once.*

A connected segment can be alternatively defined by a selection of a vertex $v \in T$ and a connected interval of children of the vertex $v$ (we call $v$ the root of the segment), so that subtrees of these children along with the edges from $v$ to these children belong to this segment. Thus, a connected segment forms a connected subtree of tree $T$.

Now let $\Delta$ be the block size – a parameter, that we will set later. We decompose the computation of $S(T)$ into $\mathcal{O}(n/\Delta)$ transitions of two types:

- **Type I:** for connected segments $T_1, T_2$ such that $|T_1| \geq \Delta$ and $|T_2| \geq \Delta$: compute $S(T_1 + T_2)$ from $S(T_1)$ and $S(T_2)$
- **Type II:** for connected segments $T_1, T_2$ such that $T_1 \subset T_2$ and $|T_2| - |T_1| = \mathcal{O}(\Delta)$: compute $S(T_2)$ from $S(T_1)$

To compute this decomposition, we will use the same algorithm as Mao did. This algorithm is included in the full version of this paper [16].

The following theorem, which corresponds to Theorem 4.5 from Mao [15], tells us how we can perform a type I transition efficiently. We prove this in Section 4.1.

▶ **Theorem 15.** *Let $A$, $B$ be $n \times n$ similarity matrices of unrooted trees. If $A$ is $t$-bounded, then $C = A \star B$ can be computed in $\mathrm{MUL}(t, n) := \widetilde{\mathcal{O}}(t^{0.8145} n^2)$ time.*

For type II transitions, in Section 4.2 we prove the following theorem, which corresponds to Theorem 4.6 from Mao [15].

▶ **Theorem 16.** *Let $T_1, T_2$ be the connected segments of tree $T$ such that $T_1 \subset T_2$ and $|T_2| - |T_1| = \mathcal{O}(\Delta)$. Given $S(T_1)$ we can compute $S(T_2)$ in $\widetilde{\mathcal{O}}(\mathrm{MUL}(\Delta, n) + n\Delta^4)$ time.*

Combining these two theorems, together with the decomposition scheme, gives us an algorithm from Theorem 1. Analysis of the time complexity of this algorithm can be found in the full version of this paper [16].

Note that in the Theorem 16 we have slightly worse time complexity for the type II transition than $\widetilde{\mathcal{O}}(\mathrm{MUL}(\Delta, n) + n\Delta^3)$ time from Mao's algorithm.

## 4.1 Type I transitions

In this section, we show how to efficiently multiply similarity matrices of unrooted trees and prove Theorem 15. Mao showed how to multiply similarity matrices of rooted trees in $\widetilde{\mathcal{O}}(t^{0.9038} n^2)$ time, where $t$ is a bound on the entries of one of the matrices. For unrooted trees, we will use his algorithm with a slight modification.

Mao's algorithm is a recursive procedure that uses the $(\max, +)$ matrix multiplication of bounded-difference matrices as a sub-procedure. Lately, Chi et al. [6] showed a better algorithm for the $(\max, +)$ product of bounded-difference matrices. This algorithm allows us to get a better exponent in Mao's matrix multiplication.

▶ **Theorem 17** ([6])**.** *There is an $\widetilde{\mathcal{O}}(n^{(3+\omega)/2}) \leq \widetilde{\mathcal{O}}(n^{2.687})$ time randomized algorithm that computes the $(\min, +)$ product of any two $n \times n$ bounded-difference matrices.*

To analyze the improvement in the exponent, we will use the following lemma implicitly proven by Mao:

▶ **Lemma 18** ([15, Section 4.4])**.** *Assume there is an $\widetilde{\mathcal{O}}(n^c)$ time algorithm that computes the $(\min, +)$ product of any two $n \times n$ bounded-difference matrices. Let $A$, $B$ be row-monotone, column-monotone and finite-upper-triangular-bounded-difference $n \times n$ matrices whose entries on the main diagonals are zero. If $A$ is $t$-bounded-upper-triangular and $\delta$ is a positive value, then $C = A \star B$ can be computed in $\widetilde{\mathcal{O}}(n^2 t^2/\delta + n^2 \delta^{c-2})$ time.*

The above complexity reaches the optimum when $n^2 t^2/\delta = n^2 \delta^{c-2}$. By setting $\delta = t^{2/(c-1)}$ we get that we can multiply the similarity matrices of rooted trees in $\widetilde{\mathcal{O}}(t^{(2c-4)/(c-1)} n^2)$ time, if one of the matrices has values bounded by $t$.

Now we want to use this algorithm also for unrooted trees. However, we need to make sure that the values from invalid cells do not affect the values of valid cells. To easily handle this, before calling the algorithm from Lemma 18 we fix the values of invalid cells based on the values of valid cells.

Let $A$, $B$ be similarity matrices of unrooted trees. We want to compute $C = A \star B$. For every invalid cell $A_{i,j}$ we set new value of $A_{i,j}$ as the maximum of valid cells $A_{i',j'}$, such that $i \leq i' \leq j' \leq j$. Let $A'$ be the resulting matrix. We can easily compute this transformation in $\mathcal{O}(n^2)$ time by using simple recursive equation $A'_{i,j} = \max(A'_{i+1,j}, A'_{i,j-1})$. Similarly, we convert matrix $B$ into matrix $B'$.

The obtained matrices $A'$ and $B'$ are obviously monotone and finite-upper-triangular. To prove that these new matrices are bounded-difference let us use induction on $i + j$. For a valid cell, our inductive thesis is satisfied. Now, take any invalid cell $A'_{i,j}$. Then, by inductive hypothesis we have that $A'_{i-1,j} \in [A'_{i-1,j-1}, A'_{i-1,j-1}+2]$ and $A'_{i,j-1} \in [A'_{i-1,j-1}, A'_{i-1,j-1}+2]$. Thus, $|A'_{i,j} - A'_{i-1,j}| \leq 2$ and $|A'_{i,j} - A'_{i,j-1}| \leq 2$, so matrices $A'$ and $B'$ are bounded-difference matrices.

Let $C' = A' \star B'$. It remains to show that $C' =_{valid} C$. Let us take any valid cell $C'_{i,j}$. There is a position $k$ such that $C'_{i,j} = A'_{i,k} + B'_{k,j}$. If one of $A'_{i,k}$, $B'_{k,j}$ is an invalid cell, then the other would be equal to $-\infty$ and we would have $C'_{i,j} < 0$. But $C'_{i,j} \geq A'_{i,i} + B'_{i,j} \geq 0$, which gives us a contradiction. Thus, invalid cells of $A'$ and $B'$ have no effect on valid cells of $C'$, so $C' =_{valid} C$.

This gives us that we can compute $C = A \star B$ in $\widetilde{\mathcal{O}}(t^{(2c-4)/(c-1)}n^2)$ time assuming we can compute the $(\min, +)$ product of any two $n \times n$ bounded-difference matrices in $\widetilde{\mathcal{O}}(n^c)$ time.

Combining this with Theorem 17 we get the proof of Theorem 15.

## 4.2 Type II transitions

In this section, we prove Theorem 16. Let $T_1$, $T_2$ be connected segments of tree $T$ such that $T_1 \subset T_2$ and $|T_2| - |T_1| = \mathcal{O}(\Delta)$. We want to compute $S(T_2)$ given $S(T_1)$.

Let us consider the path $p = e_1 e_2 \dots e_k$ in tree $T$ that goes from the root of $T_2$ to the root of $T_1$. For $1 \leq i \leq k$, let $L_i$ be the subtree of tree $T_2$ consisting of siblings of edge $e_i$ that are to the left of $e_i$ and all descendants of these edges. Additionally, let $L_{k+1}$ be a subtree of tree $T_2$ consisting of descendants of edge $e_k$ that are to the left of tree $T_1$. Similarly, we denote subtrees that are to the right of path $p$ by $R_i$ for $1 \leq i \leq k+1$. See Figure 2.

For $1 \leq i \leq j \leq k+1$, by $L_{i,j}$ we denote the tree $L_i + L_{i+1} + \ldots + L_j$ and by $R_{i,j}$ we denote the tree $R_j + R_{j-1} + \ldots + R_i$.

First, for all trees $L_{i,j}$, $R_{i,j}$, we compute the similarity matrices $S(L_{i,j})$, $S(R_{i,j})$ using the $\widetilde{\mathcal{O}}(nm^2)$ time algorithm from Section 3.2. We notice that in one run of this algorithm, we can compute $S(L_{i,j})$ for a fixed $i$ and all $j \geq i$. Thus, we only need to use that algorithm $\mathcal{O}(\Delta)$ times, which in total gives us $\widetilde{\mathcal{O}}(n\Delta^3)$ time for that step.

Now, we consider two cases:

**(a)** None of the edges $e_1, \dots, e_k$ is matched to some edge of tree $Q$.

**(b)** At least one of the edges $e_1, \dots, e_k$ is matched to some edge of tree $Q$.

For case (a), we can use Theorem 15 to compute $S(L_{1,k+1}) \star S(T_1) \star S(R_{1,k+1})$ in $\mathrm{MUL}(\Delta, n)$ time. For case (b), we define a restricted version of the similarity matrix for trees $T'$, such that the root of $T'$ has only one child.

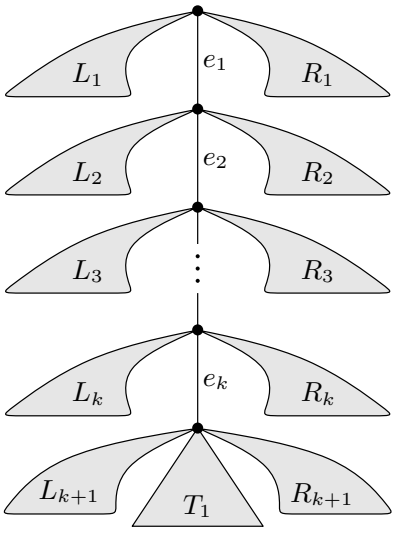



**Figure 2** Tree $T_2$. In the type II transition we compute $S(T_2)$ based on $S(T_1)$.

▶ **Definition 19** (Restricted similarity matrix). *Let $T'$ be a rooted tree such that the root $u$ of $T'$ has only one child $v$. The restricted similarity matrix $\hat{S}(T')$ is a $4|E(Q)| \times 4|E(Q)|$ matrix where:*

$$\hat{S}(T')_{i,j} = \begin{cases} \max_{e \in Q[i,j)} \{\mathrm{sim}(T' - uv, \mathrm{sub}(e) - e) + \eta(uv, e)\} & \text{if } i \leq j, j - i \leq 2|E(Q)| \\ -\infty & \text{otherwise} \end{cases}$$

We will compute $\hat{S}(\mathrm{sub}(e_i))$ for all edges $e_i$ on path $p$. But first, let us assume that we have already computed these restricted similarity matrices. We show how to compute $S(T_2)$ using these matrices.

As mentioned before we can first initialize $S(T_2)$ with $S(L_{1,k+1}) \star S(T_1) \star S(R_{1,k+1})$. Now, let us consider a single valid cell $S(T_2)_{i,j}$ that we want to compute. To cover case (b), we can iterate through:

- First edge $e_x$ from path $p$ that is matched.
- Edge $e \in Q[i,j)$ such that $e_x$ is matched to $e$.

Then, we have:

- Edges from $L_{1,x}$ are matched to edges from $Q[i, l_{i,j}(e))$ contributing $S(L_{1,x})_{i,l_{i,j}(e)}$.
- Edges from $\mathrm{sub}(e_x)$ are matched to edges from $Q[l_{i,j}(e), r_{i,j}(e)+1)$ contributing $\hat{S}(\mathrm{sub}(e_x))_{l_{i,j}(e), r_{i,j}(e)+1}$.
- Edges from $R_{1,x}$ are matched to edges from $Q[r_{i,j}(e)+1, j)$ contributing $S(R_{1,x})_{r_{i,j}(e)+1,j}$.

Note that we allow edge $e_x$ to be matched to some other edge than $e$ from $Q[l_{i,j}(e), r_{i,j}(e)+1)$. However, the cost of such matching will be at least as good as the cost of best matching that matches $e_x$ to $e$, thus it does not affect the correctness of our algorithm.

This gives us that we can compute $S(T_2)$ from all $\hat{S}(\mathrm{sub}(e_i))$ in $\widetilde{\mathcal{O}}(n^3\Delta)$. To optimize this, we can use a similar idea to the one from Algorithm 1. Instead of calculating all the cells from $S(T_2)$ separately, we can do it globally. We iterate through:

- First edge $e_x$ from path $p$ that is matched.
- Edge $e \in Q$ such that $e_x$ is matched to $e$.
- Occurrence $(I(e)_c, I(e)_{c+1})$ of the edge $e$ in the doubled Euler tour.
- Cost $a$ of matching edges from $L_{1,x}$.
- Cost $b$ of matching edges from $R_{1,x}$.

Now we can find the shortest segment $[i, j)$, such that it contains $(I(e)_c, I(e)_{c+1})$, the cost of matching $L_{1,x}$ to $Q[i, I(e)_c)$ is at least $a$ and the cost of matching $R_{1,x}$ to $Q[I(e)_{c+1} + 1, j)$ is at least $b$. Then we can use a single rangemax operation to update all valid cells $S(T_2)_{i',j'}$ for which $i' \leq i$ and $j \leq j'$. This gives us that we can compute $S(T_2)$ from $\hat{S}(\text{sub}(e_i))$'s in $\widetilde{\mathcal{O}}(n\Delta^3)$ time. The pseudocode for this part can be found in the full version of this paper [16].

Now, it remains to show how to compute all $\hat{S}(\text{sub}(e_i))$'s. We compute them starting with $e_k$ and going up to $e_1$. Thus, assume that we have already computed $\hat{S}(\text{sub}(e_i))$'s for all $i$ greater than some $x$. We want to compute $\hat{S}(\text{sub}(e_x))$. Let us consider two cases:

**(a)** None of the edges $e_{x+1}, \ldots, e_k$ is matched.

**(b)** At least one of the edges $e_{x+1}, \ldots, e_k$ is matched.

Both of these cases can be computed using similar ideas that we used to compute $S(T_2)$ from all $\hat{S}(\text{sub}(e_i))$. In the full version of this paper [16], we present how to compute case **(a)** in $\widetilde{\mathcal{O}}(n\Delta^2)$ time and case **(b)** in $\widetilde{\mathcal{O}}(n\Delta^3)$ time.

Thus, we can compute single $\hat{S}(\text{sub}(e_x))$ in $\widetilde{\mathcal{O}}(n\Delta^3)$ time and all of them in $\widetilde{\mathcal{O}}(n\Delta^4)$ time. Therefore, a whole single transition of type II can be computed in $\widetilde{\mathcal{O}}(\text{MUL}(\Delta, n) + n\Delta^4)$ time, which finishes the proof of Theorem 16.

## 5 Final remarks

We have presented the first truly subcubic algorithm for the unweighted variant of the unrooted tree edit distance. However, as mentioned before our algorithm has a slightly worse exponent than the best-known algorithm for rooted trees. The difference comes from Theorem 16 where we have time complexity $\widetilde{\mathcal{O}}(\text{MUL}(\Delta, n) + n\Delta^4)$, while Mao has $\widetilde{\mathcal{O}}(\text{MUL}(\Delta, n) + n\Delta^3)$ in corresponding theorem. It will be interesting to see if our algorithm could be optimized to match the time complexity of Mao's algorithm.

For the weighted tree edit distance probably the most interesting open problem is a question whether there exists a weakly subcubic algorithm for that version. One of the lower bounds [2] that claims this problem cannot be solved in a truly subcubic time is based on APSP conjecture. However, for APSP weakly subcubic time algorithms are already known. Williams [20] showed that the APSP problem can be solved in $\mathcal{O}(n^3/2^{\Omega(\sqrt{\log n})})$ time. Thus, we can try to obtain a weakly subcubic algorithm for the weighted tree edit distance by showing a reduction to the APSP problem.

## References


1 J. Bellando and R. Kothari. Region-based modeling and tree edit distance as a basis for gesture recognition. In *Proceedings 10th International Conference on Image Analysis and Processing*, pages 698–703, 1999. doi:10.1109/ICIAP.1999.797676.

2 Karl Bringmann, Paweł Gawrychowski, Shay Mozes, and Oren Weimann. Tree edit distance cannot be computed in strongly subcubic time (unless apsp can). *ACM Trans. Algorithms*, 16(4), July 2020. doi:10.1145/3381878.

3 Peter Buneman, Martin Grohe, and Christoph Koch. Path queries on compressed xml. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 141–152. Morgan Kaufmann, 2003. doi:10.1016/B978-012722442-8/50021-5.

4 Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 90–101, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. doi:10.5555/645925.671669.

5 Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *J. Algorithms*, 40(2):135–158, 2001. doi:10.1006/jagm.2001.1170.

6 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1529–1542, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520057.

7 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1), December 2010. doi:10.1145/1644015.1644017.

8 Bartlomiej Dudek and Pawel Gawrychowski. Edit Distance between Unrooted Trees in Cubic Time. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.45.

9 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Inf. Process. Lett.*, 181:106358, 2023. doi:10.1016/j.ipl.2023.106358.

10 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and Senthilmurugan Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57, November 2009. doi:10.1145/1613676.1613680.

11 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in rna secondary structures. In *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pages 159–168, 2003. doi:10.1109/CSB.2003.1227315.

12 Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Ben Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 696–704, USA, 2000. Society for Industrial and Applied Mathematics. doi:10.5555/338219.338628.

13 Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, volume 1461 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 1998. doi:10.1007/3-540-68530-8_8.

14 Philip N. Klein, Thomas B. Sebastian, and Benjamin B. Kimia. Shape matching using edit-distance: An implementation. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 781–790, USA, 2001. Society for Industrial and Applied Mathematics. doi:10.5555/365411.365779.

15 Xiao Mao. Breaking the cubic barrier for (unweighted) tree edit distance. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 792–803. IEEE, 2021. doi:10.1109/FOCS52979.2021.00082.

16 Krzysztof Pióro. Subcubic algorithm for (unweighted) unrooted tree edit distance, 2023. arXiv:2304.08632.

17 Thomas Sebastian, Philip Klein, and Benjamin Kimia. Recognition of shapes by editing shock graphs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26:550–571, June 2004. doi:10.1109/TPAMI.2004.1273924.

18 Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics*, 6(4):309–318, October 1990. doi:10.1093/bioinformatics/6.4.309.

19 Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, July 1979. doi:10.1145/322139.322143.

20 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 664–673, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591811.

21 Kaizhong Zhang and Dennis E. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989. doi:10.1137/0218082.

# Linear Time Construction of Cover Suffix Tree and Applications

**Jakub Radoszewski**
University of Warsaw, Poland
Samsung R&D, Warsaw, Poland

## Abstract

The Cover Suffix Tree (CST) of a string $T$ is the suffix tree of $T$ with additional explicit nodes corresponding to halves of square substrings of $T$. In the CST an explicit node corresponding to a substring $C$ of $T$ is annotated with two numbers: the number of non-overlapping consecutive occurrences of $C$ and the total number of positions in $T$ that are covered by occurrences of $C$ in $T$. Kociumaka et al. (*Algorithmica*, 2015) have shown how to compute the CST of a length-$n$ string in $\mathcal{O}(n \log n)$ time. We give an algorithm that computes the same data structure in $\mathcal{O}(n)$ time assuming that $T$ is over an integer alphabet and discuss its implications.

A string $C$ is a cover of text $T$ if occurrences of $C$ in $T$ cover all positions of $T$; $C$ is a seed of $T$ if occurrences and overhangs (i.e., prefix-suffix occurrences) of $C$ in $T$ cover all positions of $T$. An $\alpha$-partial cover ($\alpha$-partial seed) of text $T$ is a string $C$ whose occurrences in $T$ (occurrences and overhangs in $T$, respectively) cover at least $\alpha$ positions of $T$. Kociumaka et al. (*Algorithmica*, 2015; *Theor. Comput. Sci.*, 2018) have shown that knowing the CST of a length-$n$ string $T$, one can compute a linear-sized representation of all seeds of $T$ as well as all shortest $\alpha$-partial covers and seeds in $T$ for a given $\alpha$ in $\mathcal{O}(n)$ time. Thus our result implies linear-time algorithms computing these notions of quasiperiodicity. The resulting algorithm computing seeds is substantially different from the previous one (Kociumaka et al., SODA 2012, *ACM Trans. Algorithms*, 2020); in particular, it is non-recursive. Kociumaka et al. (*Algorithmica*, 2015) proposed an $\mathcal{O}(n \log n)$-time algorithm for computing a shortest $\alpha$-partial cover for each $\alpha = 1, \ldots, n$; we improve this complexity to $\mathcal{O}(n)$.

Our results are based on a new combinatorial characterization of consecutive overlapping occurrences of a substring $S$ of $T$ in terms of the set of runs (see Kolpakov and Kucherov, FOCS 1999) in $T$. This new insight also leads to an $\mathcal{O}(n)$-sized index for reporting overlapping consecutive occurrences of a given pattern $P$ of length $m$ in the optimal $\mathcal{O}(m + \mathsf{output})$ time, where $\mathsf{output}$ is the number of occurrences reported. In comparison, a general index for reporting bounded-gap consecutive occurrences of Navarro and Thankachan (*Theor. Comput. Sci.*, 2016) uses $\mathcal{O}(n \log n)$ space.

**2012 ACM Subject Classification** Theory of computation → Pattern matching

**Keywords and phrases** cover (quasiperiod), seed, suffix tree, run (maximal repetition)

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.89

**Related Version** *Full Version*: `https://arxiv.org/abs/2308.04289`

**Funding** *Jakub Radoszewski*: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

**Acknowledgements** The author thanks the anonymous reviewers for reading the manuscript carefully and providing several useful suggestions.

## 1 Introduction

The Cover Suffix Tree (CST, in short) of a string $T$, denoted as $CST(T)$, is the suffix tree of $T$ ($ST(T)$) augmented with additional nodes and values. For every substring $C$ of $T$, $CST(T)$ allows to efficiently compute the number of positions in $T$ that are covered by occurrences of $C$, provided that the node representing $C$ in $CST(T)$ is known. Thus the CST




is a generalization of string covers [3]. The CST of a string was introduced by Kociumaka et al. [29] for computing so-called *partial covers* of a string (see below). Other applications of the CST to the field of quasiperiodicity (see [2]) were discussed in [29, 30].

Let $n$ denote the length of a string $T$. Kociumaka et al. [29] presented an algorithm computing $CST(T)$ in $\mathcal{O}(n \log n)$ time. Our main result is an algorithm that constructs $CST(T)$ in $\mathcal{O}(n)$ time. We assume that $T$ is over an *integer alphabet* $\{0, \ldots, n^{\mathcal{O}(1)}\}$. This assumption has become a standard in suffix tree construction algorithms since the linear-time suffix tree construction algorithm of Farach [18].

In Section 1.1 we provide more details on the CST. Then in Section 1.2 we discuss applications of our result to computing various notions of quasiperiodicity and in Section 1.3 we present an application of our approach to a variant of text indexing.

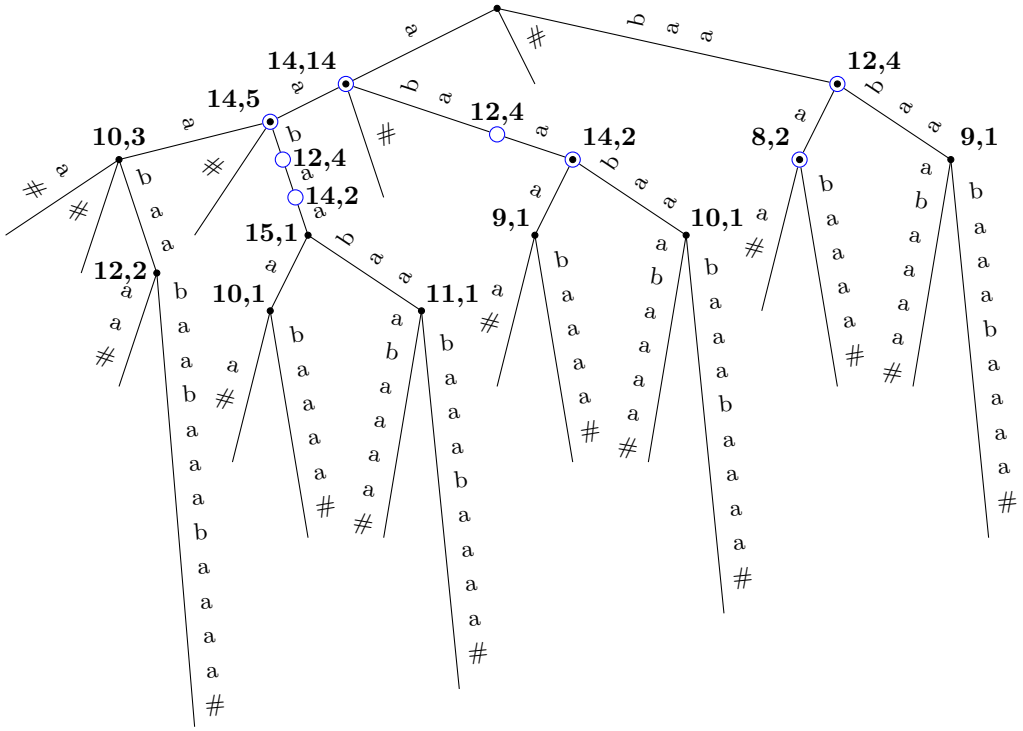



**Figure 1** $CST(T)$ for $T = \mathrm{aaabaabaabaaabaaaa\#}$. Black circles denote explicit nodes of $ST(T)$ and blue circles represent nodes corresponding to halves of square substrings in $T$. The numbers next to nodes denote $cv(v)$ and $nov(v)$.

## 1.1 Cover Suffix Tree

In the suffix tree $ST(T)$, there is a 1-to-1 correspondence between substrings of $T$ and (explicit and implicit) nodes. The same applies to $CST(T)$. The set of explicit nodes of the $CST(T)$ comprises of the explicit nodes of the suffix tree of $T$ and of nodes corresponding to halves of square substrings of $T$. Here a *square* is a string of the form $X^2 = XX$, for some string $X$ which is called the *square half*.

The CST has the same tree structure as the Maximal Augmented Suffix Tree (MAST) introduced by Apostolico and Preparata for the String Statistics Problem [4]. It was already observed by Brodal et al. [11] that the MAST of $T$ uses only $\mathcal{O}(n)$ space; this is because

the suffix tree has $\mathcal{O}(n)$ nodes [32] and the number of different square substrings of $T$ is $\mathcal{O}(n)$ [20]. By a recent result of Brlek and Li [9, 10] showing that a string of length $n$ contains at most $n$ different square substrings, it follows that the CST (and MAST) contains at most $3n$ explicit nodes.

For a node $v$ of $CST(T)$, by $\bar{v}$ we denote the substring of $T$ that corresponds to $v$. A *consecutive occurrence* of string $S$ in $T$ is a pair of indices $(i, j)$ in $T$ such that $j > i$, $S$ has occurrences starting at positions $i$ and $j$ in $T$ and $S$ does not occur at any of the positions $i + 1, \ldots, j - 1$. A consecutive occurrence $(i, j)$ of $S$ is called *overlapping* if $j < i + |S|$ and *non-overlapping* otherwise. In $CST(T)$, each node $v$ is annotated by two values (see Figure 1):

- $cv(v)$, equal to the total number of positions in $T$ covered by occurrences of $\bar{v}$, and
- $nov(v)$, equal to one plus the number of non-overlapping consecutive occurrences of $\bar{v}$ in $T$. (Intuitively, the one corresponds to the rightmost occurrence of $\bar{v}$ in $T$.)

The key property of values $cv(v)$ and $nov(v)$ is that if $u$ is an implicit node of $CST(T)$ that is located on an edge from an explicit node $v$ to its parent, then $cv(u)$ can be expressed in terms of $cv(v)$ and $nov(v)$ as follows: $cv(u) = cv(v) - (|\bar{v}| - |\bar{u}|)nov(v)$; see Figure 2.


**Figure 2** Left: the locus $v$ of $C = \text{abaabaa}$ in $CST(T)$ is an explicit node and we have $cv(v) = 10$, $nov(v) = 1$ (see Figure 1). Right: the locus $v'$ of the prefix $C' = \text{abaaba}$ of $C$ is an implicit node one character above $v$. We then have $cv(v') = cv(v) - nov(v) = 9$.

We obtain the following result.

▶ **Theorem 1.** *The Cover Suffix Tree (CST) of a string of length $n$ over an integer alphabet can be constructed in $\mathcal{O}(n)$ time.*

The $\mathcal{O}(n \log n)$-time algorithm from [29] for constructing $CST(T)$ processes the suffix tree of $T$ bottom-up, storing for each explicit node $v$ the set of occurrences of the corresponding substring of $T$ in an AVL tree. Its time complexity follows by using an efficient algorithm for merging AVL trees [13] (cf. [12]). We use a completely different approach, based on a new combinatorial observation that links overlapping consecutive occurrences of substrings of $T$ to runs in $T$ [31]. We show that the (multi)set of substrings whose overlapping consecutive occurrences are implied by a run has a simple, "triangular" structure. The values $ov(v)$ and $cv(v)$ for explicit nodes $v$ of $CST(v)$ are then computed in two bottom-up traversals, one in the tree of suffix links of $ST(T)$ and the other in the $CST(T)$.

## 1.2 Applications of CST to quasiperiodicity

Theorem 1 has several applications in the field of quasiperiodicity [2]. Basic notions of quasiperiodicity are covers [3], that were already mentioned before, and seeds [25]. A string $C$ is a cover of a string $T$ if each position of $T$ is inside at least one occurrence of $C$. A string $S$ is a seed of a string $T$ if it is a cover of a superstring of $T$. In other words, all positions of $T$ are covered by occurrences and overhangs of $S$, where an *overhang* is a prefix of $T$ being a suffix of $S$ or a suffix of $T$ being a prefix of $S$. A substring $C$ of $T$ is called an *$\alpha$-partial cover* of $T$ if the occurrences of $C$ in $T$ cover at least $\alpha$ positions in $T$. A substring $S$ of $T$ is called an *$\alpha$-partial seed* of $T$ if the occurrences and overhangs of $S$ in $T$ cover at least $\alpha$ positions in $T$. See Figure 3 for examples.

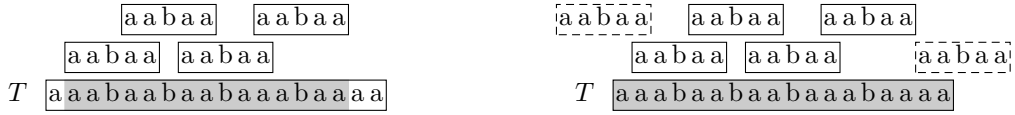



**Figure 3** $C = \mathrm{aabaa}$ is a 15-partial cover of $T$ (left). Indeed, the node $v$ of $CST(T)$ corresponding to $C$ has $cv(v) = 15$ (Figure 1). $C$ is also an 18-partial seed of $T$, hence, a seed of $T$ (right).

In [29, 30] it was noticed that given the $CST(T)$, all shortest $\alpha$-partial covers and $\alpha$-partial seeds of $T$ for a specified value of the parameter $\alpha$ can be computed in $\mathcal{O}(n)$ time. Thus our Theorem 1 implies the following result.

▶ **Corollary 2.** *Let $T$ be a string of length $n$ over an integer alphabet and $\alpha \in [1\,..\,n]$. All shortest $\alpha$-partial covers and seeds of $T$ can be computed in $\mathcal{O}(n)$ time.*

In [29] also the following problem was considered.

AllPartialCovers
**Input:** a string $T$ of length $n$
**Output:** for all $\alpha = 1, \ldots, n$, a shortest $\alpha$-partial cover of $T$

▶ **Example 3.** For $T$ from Figure 1, the solution to AllPartialCovers problem can be as follows: substring $C = \mathrm{a}$ for $\alpha \leq 14$, substring $C = \mathrm{aabaa}$ for $\alpha = 15$ (see Figure 3), and any length-$\alpha$ substring of $T$ for $\alpha \geq 16$.

An $\mathcal{O}(n \log n)$-time solution for AllPartialCovers based on $CST(T)$ and on computing the upper envelope [24] of $\mathcal{O}(n)$ line segments was presented in [29]. We obtain the following result.

▶ **Theorem 4.** *AllPartialCovers problem can be solved in $\mathcal{O}(n)$ time for a length-$n$ string over an integer alphabet.*

A linear-time algorithm computing all covers of a string was presented almost 30 years ago by Moore and Smyth [33, 34]. A rather involved linear-time algorithm computing a representation of all seeds in a string over an integer alphabet was given much more recently by Kociumaka et al. [27]. The representation (already introduced in the earlier, $\mathcal{O}(n \log n)$-time algorithm by Iliopoulos et al. [25]) consists of a set of paths in the suffix trees of $T$ and of $T$ reversed, at most one path on each edge of the suffix trees. Seeds of $T$ are exactly $|T|$-partial seeds of $T$, so Corollary 2 immediately implies an alternative linear-time algorithm computing all shortest seeds of $T$. Moreover, in [29, Theorem 3] it was observed that having $CST(T)$, the aforementioned representation of all seeds in $T$ can be computed in $\mathcal{O}(n)$ time. Thus Theorem 1 yields an alternative $\mathcal{O}(n)$-time algorithm computing the same representation of all seeds in $T$ as in [27]. The resulting algorithm is substantially different from the algorithm of [27]; in particular, it is non-recursive and arguably simpler.

Recently, Kociumaka et al. [28] showed that there exists a different representation of all seeds of a string, consisting of $\mathcal{O}(n)$ disjoint paths on just the suffix tree of $T$, and that this representation can be computed in $\mathcal{O}(n)$ time assuming an integer alphabet. This representation, however, no longer satisfies the convenient property that at most one path on each edge of the suffix tree is in the representation (see [28, Fig. 2] for an example).

## 1.3 Reporting overlapping occurrences

Data stored in the CST can be used to compute, for any substring $S$ of $T$, the number $ov(S)$ of overlapping consecutive occurrences of $S$ in $T$. We show that the technique behind Theorem 1 can be further exploited to obtain a linear-space index for reporting overlapping consecutive occurrences of a query pattern.

Navarro and Thankachan [35] proposed an index that, given a length-$m$ substring $S$ of $T$ and an interval $[\alpha, \beta]$, reports all consecutive occurrences $(i, j)$ of $S$ such that $\alpha \leq j - i \leq \beta$ in $\mathcal{O}(m + \mathsf{output})$ time, where $\mathsf{output}$ is the number of consecutive occurrences reported. The size of their index for a text $T$ of length $n$ is $\mathcal{O}(n \log n)$. We solve the following problem.

| Reporting bounded-gap overlapping consecutive occurrences |
|---|
| **Input:** A string $T$ of length $n$ |
| **Query input:** A substring $S$ of $T$, $\lvert S\rvert = m$, and a positive integer $\beta$ such that $\beta < m$ |
| **Query output:** All consecutive occurrences $(i, j)$ of $S$ in $T$ such that $j - i \leq \beta$ |

▶ **Theorem 5.** *There is an index of size $\mathcal{O}(n)$ that reports all bounded-gap overlapping consecutive occurrences of a length-$m$ pattern in $\mathcal{O}(m + \mathsf{output})$ time, where $\mathsf{output}$ is the number of consecutive occurrences reported. If $T$ is over a constant-sized alphabet, the index can be constructed in $\mathcal{O}(n)$ time. The construction time becomes expected if $T$ is over an integer alphabet.*

The data structure of Theorem 5 is superior to the data structure of [35] if $\alpha = 0$ and $\beta < m$.

In the data structure we use the same combinatorial observation as in Theorem 1. With it, a query for a pattern $S$ consists in finding the corresponding node $v$ in $CST(T)$ and reporting all "triangular" structures implied by runs that contain $v$. To this end, range minimum query data structures [8] are used to store the "bottom sides" of the triangles. Thanks to this, it is actually sufficient to store $ST(T)$ instead of the $CST(T)$. The expected time in the construction algorithm stems from using perfect hashing [21] to store children of a node of the suffix tree if $T$ is over a superconstant alphabet.

## 1.4 Structure of the paper

We start by recalling basic definitions related to strings and compact tries (including suffix trees). In Section 3 we present the proof of the main Theorem 1. Solution to AllPartialCovers (Theorem 4) is provided in Section 4. The data structure for reporting bounded-gap overlapping consecutive occurrences (proof of Theorem 5) is presented in the full version. We conclude in Section 5.

# 2 Preliminaries

## 2.1 Strings

By $\Sigma$ we denote the finite alphabet of all the considered strings. We assume that characters of a string $S$ are numbered 1 through $|S|$, with $S[i] \in \Sigma$ denoting the $i$th character. An integer $j \in [1\,..\,|S|]$ is called an index in $S$. A string $S[i]S[i+1]\cdots S[j]$ for any indices $i, j$ such that $i \leq j$ is called a *substring* of $S$. By $S[i\,..\,j]$ we denote a *fragment* of $S$ that can be viewed as a positioned substring $S[i]S[i+1]\cdots S[j]$ (formally, it is represented in $\mathcal{O}(1)$ space with a reference to $S$ and the interval $[i\,..\,j]$). We also denote $S[i\,..\,j-1]$ as $S[i\,..\,j)$. Two fragments

$S[i\,..\,j]$ and $S[i'\,..\,j']$ *match* (notation: $S[i\,..\,j] = S[i'\,..\,j']$) if the underlying substrings are the same. Similarly we define matching of a fragment and a substring. Two fragments $S[i\,..\,j]$ and $S[i'\,..\,j']$ are *equivalent* (notation: $S[i\,..\,j] \equiv S[i'\,..\,j']$) if $i = i'$ and $j = j'$. A string $U$ is called a prefix (suffix) of a string $S$ if $U = S[1\,..\,|U|]$ ($U = S[|S| - |U| + 1\,..\,|S|]$, respectively) and a *border* of $S$ if it is both a prefix and a suffix of $S$.

Henceforth by $T$ we denote the text string and by $n$ we denote $|T|$. We say that a string $S$ occurs in the text $T$ at position $i$ if $S = T[i\,..\,i + |S|)$. A pair of indices $(i, j)$ in $T$ is called a *consecutive occurrence of substring* $S$ if $i < j$, $T[i\,..\,i + |S|) = T[j\,..\,j + |S|)$ and $T[k\,..\,k + |S|) \neq S$ for all $k \in (i\,..\,j)$. A consecutive occurrence is called *overlapping* if $j < i + |S|$ and otherwise it is called *non-overlapping*. By $OvOcc(S)$ we denote the set of overlapping consecutive occurrences of $S$ in $T$.

For a string $U$ and $d \in \mathbb{Z}_{\geq 0}$, by $U^d$ we denote the $d$th power of $U$, equal to a concatenation of $d$ copies of $U$. A string $V$ is *primitive* if $V = U^d$ for $d \in \mathbb{Z}_+$ implies that $d = 1$. The following property of primitive strings is a known consequence of Fine and Wilf's lemma [19].

▶ **Lemma 6** (Synchronization property, see [14]). *A string $V$ is primitive if and only if $V$ has exactly two occurrences in $V^2$.*

A string of the form $U^2$ is called a *square*.

▶ **Theorem 7** ([20] and [15, 6]). *The number of distinct square substrings in a length-$n$ string is $\mathcal{O}(n)$ and they can all be computed in $\mathcal{O}(n)$ time assuming an integer alphabet.*

We say that a string $S$ has a period $p$ if $S[i] = S[i + p]$ holds for all $i \in [1\,..\,|S| - p]$; equivalently, if $S$ has a border of length $|S| - p$. By $per(S)$ we denote the smallest period of $S$.

A *run* in a string $T$ is a triad $(a, b, p)$ such that (1) $p$ is the smallest period of $T[a\,..\,b]$, (2) $2p \leq b - a + 1$, (3) $a = 1$ or $T[a-1] \neq T[a-1+p]$, and (4) $b = |T|$ or $T[b+1] \neq T[b+1-p]$. The *exponent* of a run $R = (a, b, p)$ is defined as $exp(R) = (b - a + 1)/p$. By $\mathcal{R}(T)$ we denote the set of all runs in $T$.

▶ **Theorem 8** ([5]). *A string $T$ of length $n$ has at most $n$ runs and they can be computed in $\mathcal{O}(n)$ time if $T$ is over an integer alphabet.*

An earlier bound $|\mathcal{R}(T)| = \mathcal{O}(n)$ together with an $\mathcal{O}(n)$-time algorithm for computing $\mathcal{R}(T)$ was proposed in [31]. All runs can be computed in $\mathcal{O}(n)$ time also for a string over a general ordered alphabet [17].

## 2.2 Compact tries

The *suffix trie* of a string $T$ contains a node for every distinct substring of $T\#$, where $\# \notin \Sigma$ is a special end marker. The root node is the empty string. For each pair of substrings $(S, Sc)$ of $T$, where $c \in \Sigma$, there is an edge from $S$ to $Sc$ labeled with the character $c$. Each suffix of $T\#$ corresponds to a leaf of the suffix trie.

A *compact suffix trie* of $T$ contains the root, the branching nodes, the leaf nodes, and possibly some other nodes of the suffix trie as *explicit nodes*. Maximal paths that do not contain explicit nodes are replaced by single compact edges, and a fragment of $T$ is used to represent the label of every such edge in $\mathcal{O}(1)$ space. The nodes that are dissolved due to compactification are called *implicit nodes*; an implicit node $u$ can be referred to as a pair $(v, d)$ where $v$ is the nearest explicit descendant of $u$ and $d$ is the distance (number of characters) between $u$ and $v$. The most common example of a compact suffix trie of $T$ is the *suffix tree* of $T$, denoted here as $ST(T)$, in which each maximal branchless path from the suffix trie is replaced by a single compact edge.

▶ **Theorem 9** ([18, 26]). *The suffix tree of a string of length $n$ over an integer alphabet can be constructed in $\mathcal{O}(n)$ time.*

For a node $v$ of a compact suffix trie $\mathcal{T}$ of $T$, the corresponding substring $\bar{v}$ of $T$ is called the *string label* of $v$. Conversely, for a substring (or fragment) $S$ of $T$, its locus in $\mathcal{T}$ is the (explicit or implicit) node $v$ of $\mathcal{T}$ such that $\bar{v} = S$.

The locus in $ST(T)$ of a substring $S$ is denoted as $locus(S)$. For a non-root explicit node $v$ of $ST(T)$, its *suffix link* leads from $v$ to the node $suf(v) = locus(X)$, where $\bar{v} = cX$, $c \in \Sigma$; it is known that $suf(v)$ is then an explicit node. By $ST'(T)$ we denote tree of suffix links in $ST(T)$. The nodes of $ST'(T)$ are the explicit nodes of $ST(T)$ and for each non-root explicit node $v$ of $ST(T)$, in $ST'(T)$ there is an edge connecting node $v$ with node $suf(v)$.

Let $Sq(T) = \{S : S^2 \text{ is a substring of } T\}$. Then the *tree structure* of $CST(T)$ is a compact suffix trie of $T$ that could be obtained from the suffix tree $ST(T)$ by making loci of substrings $Sq(T)$ explicit.

A *weighted ancestor query* on a compact suffix trie $\mathcal{T}$ is given a leaf $\ell$ of $\mathcal{T}$ and a non-negative integer $d$ and asks for the topmost (explicit) ancestor $w$ of $\ell$ such that $|\bar{w}| \geq d$. We denote such a query and its result as $w = WA(\ell, d)$. We use the following offline solution to the problem of answering $WA$ queries.

▶ **Theorem 10** ([28]). *Any $q$ weighted ancestor queries on a compact suffix trie with $\mathcal{O}(n)$ nodes of a length-$n$ string can be answered in $\mathcal{O}(n+q)$ time.*

Data structures for answering weighted ancestor queries with different complexities are known [1, 22], also in the special case of the compact suffix trie being the suffix tree [7, 23].

Let $v$ be a node of a compact suffix trie and $S = \bar{v}$. Then $cv(v)$ is formally defined as

$$cv(v) = \bigcup \{ [i \,..\, i + |S|) : T[i \,..\, i + |S|) = S \}.$$

Moreover, $nov(v)$ equals one plus the number of non-overlapping consecutive occurrences of $S$ in $T$. $CST(T)$ stores the values $cv(v)$ and $nov(v)$ for each explicit node $v$. By $occ(v)$ we further denote the total number of occurrences of $\bar{v}$ in $T$. The values $occ(v)$ for all explicit nodes of a compact suffix trie can be computed bottom-up in linear time, as $occ(v)$ is the number of leaves in the subtree of node $v$. By $ov(v)$ we denote the number of overlapping consecutive occurrences of $\bar{v}$ in $T$. We have $ov(v) + nov(v) = occ(v)$. We use the notations $cv()$, $nov()$, $ov()$ and $occ()$ also for substrings of $T$.

# 3 Construction of the CST

## 3.1 Computing the tree structure

▶ **Lemma 11.** *The tree structure of the CST of a string $T$ of length $n$ over an integer alphabet can be computed in $\mathcal{O}(n)$ time.*

**Proof.** The suffix tree of a string over an integer alphabet can be constructed in $\mathcal{O}(n)$ time (Theorem 9). By Theorem 7, the set $Sq(T)$ of square substring halves, each represented as a fragment of $T$, can be computed in $\mathcal{O}(n)$ time.

The final step is to make all implicit nodes of the suffix tree that correspond to elements of $Sq(T)$ explicit. Let $T[i \,..\, i + 2d)$ be a square substring and $\ell$ be the leaf of the suffix tree of $T$ corresponding to the suffix $T[i \,..\, n]$. Using a weighted ancestor query we can compute a pair $(v, p)$ where $v = WA(\ell, d)$ is the nearest explicit descendant of the locus $u$ of $T[i \,..\, i + d)$ and $p$ is the distance between $u$ and $v$. With Theorem 10 a batch of $\mathcal{O}(n)$ such queries

can be answered in $\mathcal{O}(n)$ time. Finally, we use Radix Sort to sort the pairs $(v, p)$ (under an arbitrary, fixed order on nodes of the suffix tree) in $\mathcal{O}(n)$ time. As a result, the loci of substrings in $Sq(T)$ are grouped by their nearest explicit descendants, and each group is sorted by decreasing depths. This allows to make all the desired implicit nodes explicit in $\mathcal{O}(n)$ time. ◀

## 3.2 Properties of overlapping consecutive occurrences

If a substring $S$ of $T$ does not have overlapping occurrences in $T$, i.e., $ov(S) = 0$, then $cv(S) = nov(S) \cdot |S| = occ(S) \cdot |S|$ is easy to compute. Hence, below we characterize overlapping consecutive occurrences of substrings. To this end, we use runs.

For indices $1 \le i \le j_1 \le j_2 \le n$, we denote the set of fragments corresponding to a path in the suffix tree of $T$:

$$\mathsf{Path}(i, j_1, j_2) = \{T[i\,..\,j] \,:\, j \in [j_1\,..\,j_2]\}.$$

For a run $R = (a, b, p)$ in $T$, we denote

$$\mathsf{Triangle}(R) = \mathsf{Triangle}(a, b, p) = \bigcup_{i=a}^{b-2p} \mathsf{Path}(i, i+p, b-p).$$

Figure 4 gives a graphical motivation for the name of this set of fragments, whereas Figure 5 shows that in some cases the triangle is "wrapped".

The following key combinatorial lemma shows that sets $\mathsf{Triangle}(R)$ for $R \in \mathcal{R}(T)$ are sufficient for counting overlapping consecutive occurrences.

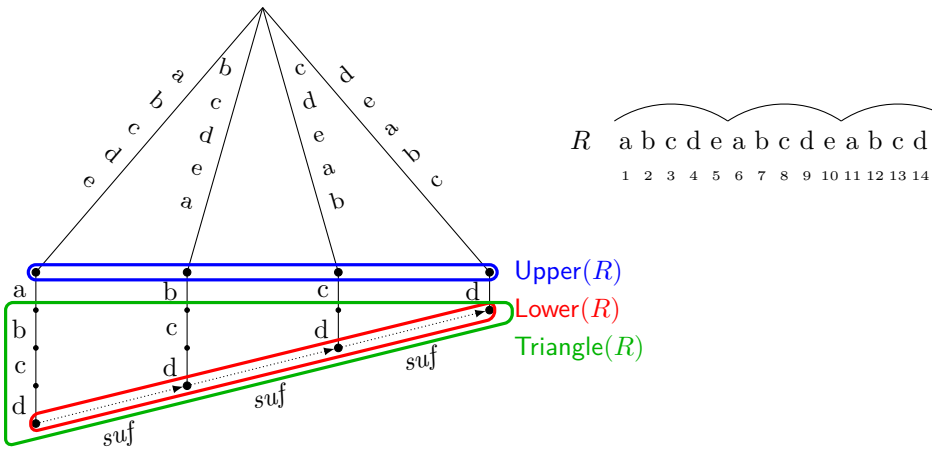



**Figure 4** Illustration of the sets $\mathsf{Triangle}(R)$, $\mathsf{Upper}(R)$ and $\mathsf{Lower}(R)$ on paths in $CST(T)$ for an example run $R = (1, 14, 5)$. All nodes representing fragments from $\mathsf{Triangle}(R)$ are distinct because $exp(R) = 2.8 \le 3$. The nodes in $\mathsf{Upper}(R)$ and $\mathsf{Lower}(R)$ are explicit in $CST(T)$ (see Observation 13).

▶ **Lemma 12.** *Let $S$ be a string of length $d$. Then $S$ has an overlapping consecutive occurrence $(i, j)$ in $T$ for some indices $i$, $j$ if and only if $S$ matches a fragment $T[i\,..\,i+d) \in \mathsf{Triangle}(R)$ for some run $R$ with period $j - i$ in $T$.*

**Proof.** ($\Rightarrow$) If $S$ has an overlapping consecutive occurrence $(i, j)$, then the substring $F = T[i\,..\,j+d)$ has a border $S$, so $F$ has a period $p = j - i < d$.

We further have $|F| = j + d - i > 2(j - i) = 2p$. Period $p$ is the smallest period of $F$; indeed, a period $q \in [1\,..\,p)$ would imply an occurrence $T[i + q\,..\,i + q + d)$ of $S$ at position $i + q$ such that $i < i + q < i + p = j - i$, so $(i, j)$ would not be a consecutive occurrence of $S$.

Finally, fragment $F$ extends to a unique maximal periodic fragment with smallest period $p$; it is a run $R = (a, b, p)$ in $T$. We have $T[i\,..\,i + d) \in \mathsf{Path}(i, i + p, b - p) \subseteq \mathsf{Triangle}(R)$ as $i \in [a\,..\,b - 2p]$.

($\Leftarrow$) Assume that $T[i\,..\,i + d) \in \mathsf{Triangle}(R)$ holds for some run $R = (a, b, p)$ and $S = T[i\,..\,i + d)$. Then $[i\,..\,i + d + p) \subseteq [a\,..\,b]$, so the period of the run implies that $T[i + p\,..\,i + p + d) = T[i\,..\,i + d) = S$. Moreover, $d > p$ by the definition of $\mathsf{Triangle}(R)$, so the two occurrences of $S$ overlap.

Finally, we need to show that $(i, j)$, for $j = i + p$, is a consecutive occurrence of $S$. If there was an occurrence $T[k\,..\,k + d) = S$ with $i < k < j$, then the string $X = T[i\,..\,i + p)$ would have an occurrence in $T[i\,..\,i + 2p) = X^2$ being neither a prefix nor a suffix of $X^2$. String $X$ is primitive, as otherwise the run $R$ would have a period smaller than $p$. Therefore this situation is impossible by the synchronization property (Lemma 6). ◀

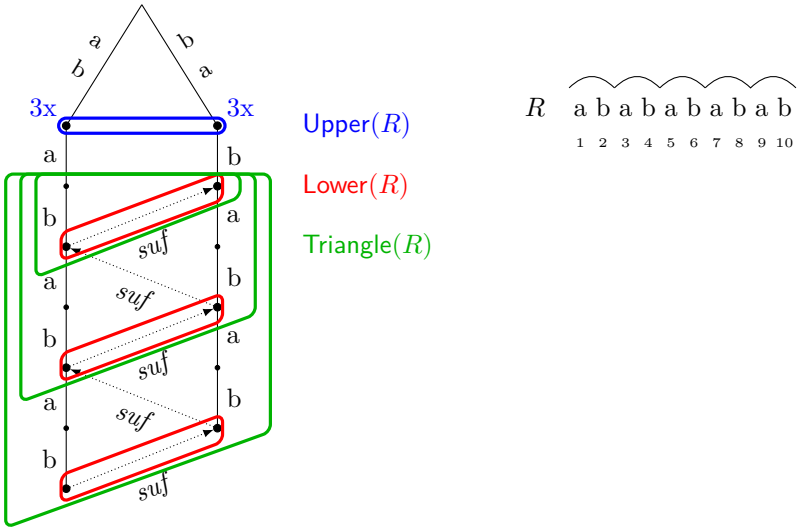



**Figure 5** Illustration of the sets $\mathsf{Triangle}(R)$, $\mathsf{Upper}(R)$ and $\mathsf{Lower}(R)$ on paths in $CST(T)$ for a run $R = (1, 10, 2)$ with exponent 5. The substrings in the set $\mathsf{Triangle}(R)$ form a multiset being the sum of the two trapezia and a triangle. The set $\mathsf{Upper}(R)$ contains six fragments; three of them match substring ab, and the remaining three match ba.

For a run $R = (a, b, p)$, we further denote:

$$\mathsf{Upper}(a, b, p) = \{T[i\,..\,i + p) \,:\, i \in [a\,..\,b - 2p]\}$$
$$\mathsf{Lower}(a, b, p) = \{T[i\,..\,b - p] \,:\, i \in [a\,..\,b - 2p]\}$$

Intuitively, $\mathsf{Lower}(R)$ consists of bottommost endpoints of paths $\mathsf{Path}$ from $\mathsf{Triangle}(R)$ and $\mathsf{Upper}(R)$ consists of parents of topmost endpoints of these paths. Informally, they are the "lower side" and the "excluded upper side" of the triangle; see also Figures 4 and 5. Below we show basic properties of these sets.

▶ **Observation 13.** *Let $R$ be a run in $T$.*

**(a)** *All fragments in $\mathsf{Upper}(R)$ are square halves in $T$.*

**(b)** *The loci of fragments in $\mathsf{Lower}(R)$ are explicit nodes in $ST(T)$.*

**Proof.** (a) By the periodicity of run $R$, each fragment $T[i\,..\,i+p) \in \mathsf{Upper}(R)$ is followed by a matching fragment $T[i+p\,..\,i+2p)$. This is because $i \le b-2p$. Hence, $T[i\,..\,i+2p)$ is indeed a square in $T$.

(b) Let $T[i\,..\,b-p] \in \mathsf{Lower}(R)$ and $c = T[b+1-p]$. The period of the run implies that $T[i\,..\,b-p] = T[i+p\,..\,b]$.

If $T[i\,..\,b-p]$ is a suffix of $T$, its locus in $ST(T)$ is explicit as the locus has children along the characters $c$ and $\#$.

Otherwise, character $c' = T[b+1]$ is different from $c$ by the right maximality of the run $R$. Hence, $T[i\,..\,b-p]c$ and $T[i\,..\,b-p]c'$ are different substrings of $T$, as claimed. ◀

Let us note that if $exp(R) > 3$, then each of the sets $\mathsf{Upper}(R)$, $\mathsf{Triangle}(R)$ may contain matching fragments; see Figure 5.

## 3.3 Counting overlapping consecutive occurrences

For each explicit node $v$ of $CST(T)$, instead of $nov(v)$, we will compute the number $ov(v)$ of overlapping consecutive occurrences of the substring $\bar{v}$ in $T$.

For a set $\mathcal{F}$ of fragments of $T$, we denote by $\#_v(\mathcal{F}) = |\{T[i\,..\,j] \in \mathcal{F} : \bar{v} = T[i\,..\,j]\}|$ the number of fragments in $\mathcal{F}$ that match $\bar{v}$. Lemma 12 implies the following formula for $ov(v)$.

▶ **Observation 14.** *For a node $v$ of $CST(T)$, $ov(v) = \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Triangle}(R))$.*

We will show how to efficiently evaluate these formulas for all explicit nodes $v$ simultaneously.

For each explicit node $v$ of $CST(T)$ we will compute two counters:

$$C_{upper}[v] = \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Upper}(R)), \quad C_{lower}[v] = \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Lower}(R)).$$

That is, $C_{upper}[v]$ ($C_{lower}[v]$) stores the number of times fragments matching the substring $\bar{v}$ occur in $\mathsf{Upper}(R)$ ($\mathsf{Lower}(R)$, respectively) over all runs $R \in \mathcal{R}(T)$.

For an explicit node $v$ of $CST(T)$, by $subtree(v)$ we denote the set of explicit descendants of $v$ in the tree (including $v$). The following lemma shows how to compute $ov$ from the counters $C_{upper}$ and $C_{lower}$. The lemma follows by Observation 14.

▶ **Lemma 15.** *For an explicit node $v$ of the $CST(T)$, we have*

$$ov(v) = \sum_{w\in subtree(v)} (C_{lower}[w] - C_{upper}[w]).$$

**Proof.** If node $x$ is an ancestor of node $y$, by $x \rightsquigarrow y$ we denote the set of explicit nodes on the path from $x$ to $y$. By *root* we denote the root of $CST(T)$. By the definitions of $\mathsf{Triangle}(R)$, $\mathsf{Upper}(R)$ and $\mathsf{Lower}(R)$ and Observation 14, we have:

$$\begin{aligned} ov(v) &= \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Triangle}(R)) \\ &= \sum_{(a,p,b)\in\mathcal{R}(T)} \sum_{i=a}^{b-2p} \#_v(\mathsf{Path}(i, i+p, b-p)) \\ &= \sum_{(a,p,b)\in\mathcal{R}(T)} |\{i \in [a\,..\,b-2p] : v \in (locus(T[i\,..\,i+p]) \rightsquigarrow locus(T[i\,..\,b-p]))\}| \end{aligned}$$

$$= \sum_{(a,p,b)\in\mathcal{R}(T)} |\{i \in [a\,..\,b-2p] \,:\, v \in (root \rightsquigarrow locus(T[i\,..\,b-p]))\}|$$
$$- \sum_{(a,p,b)\in\mathcal{R}(T)} |\{i \in [a\,..\,b-2p] \,:\, v \in (root \rightsquigarrow locus(T[i\,..\,i+p))\}|$$
$$= \sum_{R\in\mathcal{R}(T)} \sum_{w\in subtree(v)} \#_w(\mathsf{Lower}(R)) - \sum_{R\in\mathcal{R}(T)} \sum_{w\in subtree(v)} \#_w(\mathsf{Upper}(R))$$
$$= \sum_{w\in subtree(v)} (C_{lower}[w] - C_{upper}[w]).$$ ◀

### 3.3.1 Computing $C_{lower}$ and $C_{upper}$

Let us recall that $ST'(T)$ is the tree of suffix links of $ST(T)$.

▶ **Observation 16.** *For each run $R$ in $T$, $\mathsf{Lower}(R)$ forms a path in $ST'(T)$.*

▶ **Lemma 17.** *The counters $C_{lower}[v]$ for all explicit nodes $v$ of $CST(T)$ can be computed in $\mathcal{O}(n)$ time.*

**Proof.** By the observation, $C_{lower}[v]$ is simply the number of paths $\mathsf{Lower}(R)$ that cover node $v$ in $ST'(T)$ (in particular, no two fragments in a single set $\mathsf{Lower}(R)$ match).

To count paths covering each node in a rooted tree we apply a standard approach using $\pm1$ counters. Initially all counters $C_{lower}[v]$ are equal to 0. For each run $R = (a, b, p) \in \mathcal{R}(T)$, we increment $C_{lower}[v]$ for the bottom endpoint $v$ of the path $\mathsf{Lower}(R)$ ($v = locus(T[a\,..\,b-p])$) and decrement $C_{lower}[u]$ for the parent $u$ in $ST'(T)$ of the top endpoint of $\mathsf{Lower}(R)$ ($u = locus(T[b-2p+1\,..\,b-p])$). In the end for each node $u$ of $ST'(T)$ in a bottom-up order, we add $C_{lower}[v]$ to $C_{lower}[u]$ for all children $v$ of $u$ in $ST'(T)$.

Let us summarize and analyze the complexity of the algorithm. Tree $ST'(T)$ has $\mathcal{O}(n)$ nodes. By Theorem 8, there are at most $n$ paths $\mathsf{Lower}(R)$ and all runs $R$ can be computed in $\mathcal{O}(n)$ time. The endpoints of all paths $\mathsf{Lower}(R)$ can be located in $ST'(T)$ in $\mathcal{O}(n)$ time using weighted ancestor queries in $ST(T)$ (Theorem 10). Finally, the bottom-up traversal of the tree $ST'(T)$ takes $\mathcal{O}(n)$ time. ◀

We proceed to computing counters $C_{upper}$. Let us define an operation *rot* such that $rot(cX) = Xc$ for a string $X$ and character $c \in \Sigma$. For $k \in \mathbb{Z}_{\geq 0}$, by $rot^k(S)$ we denote the composition of *rot* $k$ times. If $S' = rot^k(S)$ for some strings $S, S'$ and $k \in \mathbb{Z}_{\geq 0}$, we say that $S'$ is a *cyclic rotation* of $S$. We also say that $S$ and $S'$ are *cyclically equivalent*.

For each run $R$ in $T$, the strings in $\mathsf{Upper}(R)$ are cyclic rotations of each other. This motivates introduction of the following directed graph $G = (V, E)$. The set of vertices is $V = Sq(T)$ and the arcs are defined as follows: $(S, S') \in E$ if and only if $S, S' \in V$ and $S' = rot(S)$. Instead of addressing vertices of $G$ by substrings of $T$, we will address them by their loci in $CST(T)$ which are explicit nodes of $CST(V)$.

▶ **Observation 18.** *For each run $R$ in $T$, $\mathsf{Upper}(R)$ corresponds to a (directed) walk in $G$.*

Let us note that the vertices (and arcs) on the walk $\mathsf{Upper}(R)$ may repeat if $exp(R) > 3$ (see Figure 5 again). In particular, in this case $\mathsf{Upper}(R)$ is contained in a cycle in $G$.

We proceed to the construction of graph $G$. More precisely, a sufficient subset of arcs of $G$ is constructed.

▶ **Lemma 19.** *A subset $E'$ of $E$ containing all arcs that belong to any walk $\mathsf{Upper}(R)$, for $R \in \mathcal{R}(T)$, can be constructed in $\mathcal{O}(n)$ time.*

**Proof.** For each distinct square substring $T[i\,..\,i+2d)$ of $T$, we insert into $E'$ an arc from $\mathit{locus}(T[i\,..\,i+d))$ to $\mathit{locus}(T[i+1\,..\,i+d])$ if $T[i+1\,..\,i+d] \in Sq(T)$; the latter condition can be checked from the construction of the tree structure of $CST(T)$ (Lemma 11). By Theorem 7, square substrings of $T$ can be enumerated in $\mathcal{O}(n)$ time. Then we use off-line weighted ancestor queries (Theorem 10) on $CST(T)$ to find the desired loci. This concludes that the time complexity is $\mathcal{O}(n)$. Now let us argue for the correctness of this algorithm in two steps.

Why $E' \subseteq E$: When adding an arc from $\mathit{locus}(T[i\,..\,i+d))$ to $\mathit{locus}(T[i+1\,..\,i+d])$, we know that $T[i\,..\,i+d) \in Sq(T)$ and we check if $T[i+1\,..\,i+d] \in Sq(T)$. Hence, an arc connects two vertices of $V = Sq(T)$. Finally, we have $\mathit{rot}(T[i\,..\,i+d)) = T[i+1\,..\,i+d]$ because $T[i\,..\,i+2d)$ is a square. Consequently, $E' \subseteq E$.

Why all arcs that belong to any walk $\mathsf{Upper}(R)$ are in $E'$: Let $T[i\,..\,i+p), T[i+1\,..\,i+p] \in \mathsf{Upper}(a,b,p)$ be two consecutive elements. Then $a \le i < b-2p$, so $T[i\,..\,i+2p)$ is a square. Therefore, $(\mathit{locus}(T[i\,..\,i+p)), \mathit{locus}(T[i+1\,..\,i+p])) \in E'$ by definition. ◀

In the lemma above, it can be the case that $E' \subsetneq E$ if there are two substrings $S, S' \in Sq(T)$ such that $S' = \mathit{rot}(S)$ but there are no two fragments $T[i\,..\,i+|S|)$, $T[i+1\,..\,i+1+|S|)$ matching $S$ and $S'$, respectively. Moreover, it can happen that $E'$ contains an arc that does not belong to any walk $\mathsf{Upper}(R)$. Indeed, when an arc from $\mathit{locus}(T[i\,..\,i+d))$ to $\mathit{locus}(T[i+1\,..\,i+d])$ is added to $E'$, we avoid the unnecessary check if $T[i\,..\,i+d), T[i+1\,..\,i+d]$ belong to a set $\mathsf{Upper}(R)$ for any run $R$.

▶ **Lemma 20.** *The counters $C_{upper}[v]$ for all explicit nodes $v$ of $CST(T)$ can be computed in $\mathcal{O}(n)$ time.*

**Proof.** By Observation 18, $C_{upper}[v]$ is the total number of times that walks $\mathsf{Upper}(R)$ visit the node $v \in V$. We will be able to compute these counters efficiently using the fact that graph $G$ has a particularly simple structure: it is a collection of disjoint cycles and paths. The same applies to the graph $G' = (V, E')$ that is computed in Lemma 19. For a node $u$, by $\mathit{next}(u)$ we denote the unique node $v$ such that $(u, v) \in E'$, and $\perp$ if no such node exists.

We can find all cycles in $G'$ using the DFS. Then we apply an algorithm using $\pm 1$ counters (as in the proof of Lemma 17) and additional counters $C'$ assigned to cycles. Initially all counters are equal to 0. For each cycle $Q$, let us order the nodes $v_1, \ldots, v_{|Q|} \in Q$ along the cycle (arbitrarily) and assign them consecutive id numbers $\mathit{id}(v_i) = i$.

For each walk $\mathsf{Upper}(R)$, $R = (a, b, p) \in \mathcal{R}(T)$, we check if its endpoints $v_1$ and $v_2$ ($v_1 = \mathit{locus}(T[a\,..\,a+p))$ and $v_2 = \mathit{locus}(T[b-2p\,..\,b-p))$) belong to a cycle. If not, we increment $C_{upper}[v_1]$; we also decrement $C_{upper}[\mathit{next}(v_2)]$ if $\mathit{next}(v_2) \neq \perp$. Otherwise, if $v_1, v_2$ belong to a cycle $Q$, we increase the cycle counter $C'[Q]$ by $\lfloor\, |\mathsf{Upper}(R)| \,/\, |Q| \,\rfloor$. Moreover (for $v_1, v_2 \in Q$), if $\mathit{id}(v_1) \le \mathit{id}(v_2)$, we increment $C_{upper}[v_1]$ and decrement $C_{upper}[\mathit{next}(v_2)]$ if $\mathit{id}(v_2) < |Q|$. If, however, $\mathit{id}(v_1) > \mathit{id}(v_2)$, we increment $C_{upper}[v_1]$ and $C_{upper}(u)$ for the node $u \in Q$ with $\mathit{id}(u) = 1$ and decrement $C_{upper}[\mathit{next}(v_2)]$, thus "breaking the cyclicity".

For each cycle $Q$, let us remove the arc $(v, \mathit{next}(v))$ for vertex $v \in Q$ such that $\mathit{id}(v) = |Q|$. This way $G'$ becomes acyclic; it can be viewed as a forest in which each tree is a path. For each node $v$ of the modified graph $G'$ in topological order, we add $C_{upper}[v]$ to $C_{upper}[\mathit{next}(v)]$ if $\mathit{next}(v) \neq \perp$. Finally, for each original cycle $Q$ in $G'$, we increase $C_{upper}[v]$ for all vertices $v \in Q$ by the counter $C'[Q]$. This way we have computed all the counters $C_{upper}$ as desired.

Let us analyze the complexity. By Lemma 19, graph $G' = (V, E')$ can be constructed in $\mathcal{O}(n)$ time. By Theorem 8, there are at most $n$ paths $\mathsf{Upper}(R)$ and all runs $R$ can be computed in $\mathcal{O}(n)$ time. The endpoints of walks $\mathsf{Upper}(R)$ can be located in $G'$ in $\mathcal{O}(n)$ time using weighted ancestor queries in $CST(T)$ (Theorem 10). Finally, the computation of counters via DFS and topological ordering takes $\mathcal{O}(n)$ time. ◀

This concludes efficient computation of the numbers of overlapping and non-overlapping consecutive occurrences.

▶ **Lemma 21.** *Values $ov(v)$ and $nov(v)$ for all explicit nodes $v$ of $CST(T)$ can be computed in $\mathcal{O}(n)$ time.*

**Proof.** We compute the counters $C_{lower}$ and $C_{upper}$ using Lemmas 17 and 20, respectively. By the formula from Lemma 15, for each node $v$ of $CST(T)$ in the bottom-up order, $ov(v)$ can be computed as a sum of $C_{lower}[v] - C_{upper}[v]$ and the sum of values $ov(w)$ for all explicit children $w$ of $v$. Such values can be computed via a bottom-up traversal in $\mathcal{O}(n)$ time.

Finally we recall that $nov(v) = occ(v) - ov(v)$ and that $occ(v)$ for all explicit nodes of $CST(T)$ can be easily computed in $\mathcal{O}(n)$ time bottom-up. ◀

## 3.4 Computing coverage

For a substring $S$ of $T$, we introduce the following notations:

$$cv\_ov(S) = \sum_{(i,j)\in OvOcc(S)} (j-i), \quad cv\_nov(S) = nov(S)\cdot|S|.$$

As before, we denote $cv\_ov(v) = cv\_ov(\bar{v})$ and $cv\_nov(v) = cv\_nov(\bar{v})$ for nodes $v$ of $CST(T)$. The proof of the following observation provides intuition on these definitions.

▶ **Observation 22.** *For every substring $S$ of $T$, $cv(S) = cv\_ov(S) + cv\_nov(S)$.*

**Proof.** Let us assign each position $k$ of $T$ that is covered by an occurrence of $S$ to the rightmost occurrence $T[i\,..\,i+|S|)$ of $S$ with $i < k$. Let $j$ be the next occurrence of $S$ to the right of position $i$ (then $j > k$), if any. If $j$ exists and $(i,j) \in OvOcc(S)$, then position $k$ is counted in $cv\_ov(S)$. Otherwise position $k$ is counted in $cv\_nov(S)$. ◀

Values $cv\_nov(v)$ for explicit nodes $v$ of $CST(T)$ can be easily computed using values $nov(v)$ computed in Lemma 21. Lemma 12 yields the following formula for $cv\_ov(v)$.

▶ **Observation 23.** *For a node $v$ of $CST(T)$, $cv\_ov(v) = \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Triangle}(R))\cdot per(R)$.*

Now $cv\_ov$ values can be computed similarly as $ov$ values were computed in Section 3.3. We just need to multiply counter updates by periods of respective runs.

▶ **Lemma 24.** *The values $cv\_ov(v)$ for all explicit nodes $v$ of $CST(T)$ can be computed in $\mathcal{O}(n)$ time.*

**Proof.** Let

$$C'_{upper}[v] = \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Upper}(R))\cdot per(R), \quad C'_{lower}[v] = \sum_{R\in\mathcal{R}(T)} \#_v(\mathsf{Lower}(R))\cdot per(R).$$

Following the proof of Lemma 15 it can be readily verified that for every explicit node $v$,

$$cv\_ov(v) = \sum_{w\in subtree(v)} (C'_{lower}[w] - C'_{upper}[w]).$$

The counters $C'_{lower}[v]$ ($C'_{upper}[v]$) for all explicit nodes can be computed as in Lemma 17 (Lemma 20, respectively), where instead of $\pm1$ counters, for each path $\mathsf{Lower}(R)$ (walk $\mathsf{Upper}(R)$, respectively), $R \in \mathcal{R}(T)$, we add and subtract $per(R)$ in the respective nodes (and increase cycle counters $C'[Q]$ by amounts $\lfloor\, |\mathsf{Upper}(R)|\,/\,|Q|\,\rfloor \cdot per(R)$). ◀

This concludes the construction of $CST(T)$.

▶ **Theorem 1.** *The Cover Suffix Tree (CST) of a string of length $n$ over an integer alphabet can be constructed in $\mathcal{O}(n)$ time.*

**Proof.** Lemma 11 can be used to construct the tree structure of $CST(T)$. We compute $occ(v)$ for all explicit nodes of $CST(T)$ in a bottom-up traversal and $ov(v)$ using Lemma 21, which lets us compute $nov(v) = occ(v) - ov(v)$ for all explicit nodes. Then for all explicit nodes we compute $cv\_ov(v)$ using Lemma 24, which lets us compute $cv(v)$ for all explicit nodes using values $cv\_nov(v)$ that, in turn, depend on $nov(v)$. Each of the lemmas, as well as the bottom-up processing, requires $\mathcal{O}(n)$ time. ◀

## 4 Solution to AllPartialCovers

An $\mathcal{O}(n \log n)$-time solution to AllPartialCovers from [29] is based on computing the upper envelope of $\mathcal{O}(n)$ line segments, each connecting points $(|v|, cv(v))$ and $(|v| - k, cv(v) - k \cdot nov(v))$ constructed for an edge of $CST(T)$ from $v$ to its parent $v'$ containing $k$ implicit nodes. An upper envelope of $\mathcal{O}(n)$ line segments can be computed in $\mathcal{O}(n \log n)$ time [24].

We show that the AllPartialCovers problem can be solved in $\mathcal{O}(n)$ time using the following observation. A substring $C$ of $T$ is called *branching* if the locus of $C$ in $ST(T)$ is a branching node.

▶ **Lemma 25.** *If $C$ is a substring of $T$, then there is a substring $C'$ of $T$ such that $|C'| = |C|$, $cv(C') \geq cv(C)$ and $C'$ is branching or a suffix of $T$.*

**Proof.** Let $C_0 = C$. If $C_0$ is branching or $C_0$ is a suffix of $T$, we are done. Otherwise, all occurrences of $C_0$ in $T$ are followed by the same character. Let $a$ be this character, $X$ be $C$ without its first character, and $C_1 = Xa$. We have $|C_1| = |C_0|$ and $cv(C_1) \geq cv(C_0)$. We use this construction to obtain substrings $C_1, C_2, \ldots$ The sequence ends at the first substring that is branching or a suffix of $T$; such a substring exists since the rightmost occurrence of $C_i$ in $T$, for $i \geq 1$, is located to the right of the rightmost occurrence of $C_{i-1}$ in $T$. Let $C_k$, for $k \geq 1$, be the last substring in this sequence. By the construction, $|C_k| = |C_0| = |C|$, $cv(C_k) \geq cv(C)$ and $C_k$ is branching or a suffix of $T$. We choose $C' = C_k$. ◀

By the lemma, in the solution to AllPartialCovers it suffices to iterate over all suffixes of $T$ and branching nodes of $ST(T)$. We obtain the following result that was already stated in Section 1.

**Algorithm 1** Solution to AllPartialCovers.

```
for i := 1 to n do shortest[n − i + 1] := T[i . . n];
foreach branching node v of ST(T) do
    if |v̄| < |shortest[cv(v)]| then
        shortest[cv(v)] := v̄;
for i := n − 1 down to 1 do
    if |shortest[i]| > |shortest[i + 1]| then
        shortest[i] := shortest[i + 1];
```

▶ **Theorem 4.** *AllPartialCovers problem can be solved in $\mathcal{O}(n)$ time for a length-$n$ string over an integer alphabet.*

**Proof.** We apply Algorithm 1. Clearly, the algorithm works in $\mathcal{O}(n)$ time. Let us argue for its correctness.

In the algorithm an auxiliary array *shortest* is used that stores fragments of $T$ represented in $\mathcal{O}(1)$ space each. By Lemma 25, after the foreach-loop, $shortest[\alpha] = C$ for $\alpha \in [1\,..\,n]$ if $C$ is a shortest substring of $T$ such that $cv(C) = \alpha$. At the end, $shortest[\alpha] = C$ if $C$ is a shortest substring of $T$ such that $cv(C) \geq \alpha$. Hence, $shortest[\alpha]$ is a shortest $\alpha$-partial cover of $T$ by definition. ◀

## 5 Conclusions

We have designed the first linear-time algorithm computing the Cover Suffix Tree. We have shown several applications of this result, some of which follow directly from previous work. Experimental comparison of our algorithms for computing the Cover Suffix Tree and the set of seeds in a string with implementations of existing methods from [16] is left as future work.

It remains an open problem if our approach can help to improve upon the $\mathcal{O}(n \log n)$-time algorithm of Brodal et al. [11] for constructing MAST.

## References


1. Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Transactions on Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.
2. Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science*, 119(2):247–265, 1993. doi:10.1016/0304-3975(93)90159-Q.
3. Alberto Apostolico, Martin Farach, and Costas S. Iliopoulos. Optimal superprimitivity testing for strings. *Information Processing Letters*, 39(1):17–20, 1991. doi:10.1016/0020-0190(91)90056-N.
4. Alberto Apostolico and Franco P. Preparata. Data structures and algorithms for the string statistics problem. *Algorithmica*, 15(5):481–494, 1996. doi:10.1007/BF01955046.
5. Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
6. Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.22.
7. Djamal Belazzougui, Dmitry Kosolobov, Simon J. Puglisi, and Rajeev Raman. Weighted ancestors in suffix trees revisited. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021*, volume 191 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.8.
8. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *4th Latin American Symposium on Theoretical Informatics, LATIN 2000*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
9. Srečko Brlek and Shuo Li. On the number of squares in a finite word, 2022. arXiv:2204.10204.
10. Srečko Brlek and Shuo Li. On the number of distinct squares in finite sequences: Some old and new results. In Anna E. Frid and Robert Mercas, editors, *14th International Conference on Combinatorics on Words, WORDS 2023*, volume 13899 of *Lecture Notes in Computer Science*, pages 35–44. Springer, 2023. doi:10.1007/978-3-031-33180-0_3.

11 Gerth Stolting Brodal, Rune B. Lyngso, Anna Ostlin, and Christian N. S. Pedersen. Solving the string statistics problem in time O(n log n). In Peter Widmayer, Francisco Triguero Ruiz, Rafael Morales Bueno, Matthew Hennessy, Stephan J. Eidenbenz, and Ricardo Conejo, editors, *29th International Colloquium on Automata, Languages and Programming, ICALP 2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 728–739. Springer, 2002. doi:10.1007/3-540-45465-9_62.

12 Gerth Stolting Brodal and Christian N. S. Pedersen. Finding maximal quasiperiodicities in strings. In Raffaele Giancarlo and David Sankoff, editors, *11th Annual Symposium on Combinatorial Pattern Matching, CPM 2000*, volume 1848 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 2000. doi:10.1007/3-540-45123-4_33.

13 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *Journal of the ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.

14 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

15 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.

16 Patryk Czajka and Jakub Radoszewski. Experimental evaluation of algorithms for computing quasiperiods. *Theoretical Computer Science*, 854:17–29, 2021. doi:10.1016/j.tcs.2020.11.033.

17 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 63:1–63:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.63.

18 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.

19 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.

20 Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998. doi:10.1006/jcta.1997.2843.

21 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

22 Moses Ganardi and Paweł Gawrychowski. Pattern matching on grammar-compressed strings in linear time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2833–2846. SIAM, 2022. doi:10.1137/1.9781611977073.110.

23 Paweł Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. Weighted ancestors in suffix trees. In Andreas S. Schulz and Dorothea Wagner, editors, *22th Annual European Symposium on Algorithms, Wrocław, Poland, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 455–466. Springer, 2014. doi:10.1007/978-3-662-44777-2_38.

24 John Hershberger. Finding the upper envelope of n line segments in O(n log n) time. *Information Processing Letters*, 33(4):169–174, 1989. doi:10.1016/0020-0190(89)90136-1.

25 Costas S. Iliopoulos, Dennis W. G. Moore, and Kunsoo Park. Covering a string. *Algorithmica*, 16(3):288–297, 1996. doi:10.1007/BF01955677.

26 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

27 Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear time algorithm for seeds computation. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 1095–1112. SIAM, 2012. doi:10.1137/1.9781611973099.86.

**28** Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. A linear-time algorithm for seeds computation. *ACM Transations on Algorithms*, 16(2):27:1–27:23, 2020. doi:10.1145/3386369.

**29** Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Fast algorithm for partial covers in words. *Algorithmica*, 73(1):217–233, 2015. doi:10.1007/s00453-014-9915-3.

**30** Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient algorithms for shortest partial seeds in words. *Theoretical Computer Science*, 710:139–147, 2018. doi:10.1016/j.tcs.2016.11.035.

**31** Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.

**32** Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976. doi:10.1145/321941.321946.

**33** Dennis W. G. Moore and William F. Smyth. Computing the covers of a string in linear time. In Daniel Dominic Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA 1994*, pages 511–515. ACM/SIAM, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314636.

**34** Dennis W. G. Moore and William F. Smyth. A correction to "An optimal algorithm to compute all the covers of a string". *Information Processing Letters*, 54(2):101–103, 1995. doi:10.1016/0020-0190(94)00235-Q.

**35** Gonzalo Navarro and Sharma V. Thankachan. Reporting consecutive substring occurrences under bounded gap constraints. *Theoretical Computer Science*, 638:108–111, 2016. doi:10.1016/j.tcs.2016.02.005.

# Simultaneous Representation of Interval Graphs in the Sunflower Case

**Ignaz Rutter**
Faculty of Computer Science and Mathematics, University of Passau, Germany

**Peter Stumpf**
Faculty of Computer Science and Mathematics, University of Passau, Germany

## Abstract

A simultaneous representation of (vertex-labeled) graphs $G_1, \ldots, G_k$ consists of a (geometric) intersection representation $R_i$ for each graph $G_i$ such that each vertex $v$ is represented by the same geometric object in each $R_i$ for which $G_i$ contains $v$. While Jampani and Lubiw showed that the existence of simultaneous interval representations for $k = 2$ can be tested efficiently (2010), testing it for graphs where $k$ is part of the input is NP-complete (Bok and Jedličková, 2018). An important special case of simultaneous representations is the *sunflower case*, where $G_i \cap G_j = (V(G_i) \cap V(G_j), E(G_i) \cap E(G_j))$ is the same graph for each $i \neq j$. We give an $O(\sum_{i=1}^{k}(|V(G_i)| + |E(G_i)|))$-time algorithm for deciding the existence of a simultaneous interval representation for the sunflower case, even when $k$ is part of the input. This answers an open question of Jampani and Lubiw.

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

**Keywords and phrases** Interval Graphs, Sunflower Case, Simultaneous Representation, Recognition, Geometric Intersection Graphs

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.90

**Funding** This work was funded by grant RU 1903/3-1 of the German Research Foundation (DFG).

## 1 Introduction

For a family of geometric objects, the *intersection graph* is a graph that has for each object a vertex such that two vertices are adjacent if and only if their objects intersect. Its *representation* is the assignment of the objects to the vertices. In this paper, we consider *interval representations*, which are assignments of intervals on the real line to the vertices of a graph $G$ such that two vertices of $G$ are adjacent if and only if their intervals intersect. Graph $G$ is an *interval graph* if it has such a representation; see Figure 1.

A fundamental problem in the area of intersection graphs is the *recognition* problem, where the task is to decide whether a given graph $G$ admits a particular type of (geometric) intersection representation. The simultaneous representation problem is a generalization of the recognition problem that asks for $k$ input graphs $G_1, \ldots, G_k$ (with vertex labels) whether there exist corresponding representations $R_1, \ldots, R_k$ such that each vertex $v$ that is shared by two graphs $G_i$ and $G_j$ is represented by the same geometric object in $R_i$ and in $R_j$. For ease of notation, we refer to $\mathcal{G} = (G_1, \ldots, G_k)$ as a *simultaneous graph*, and

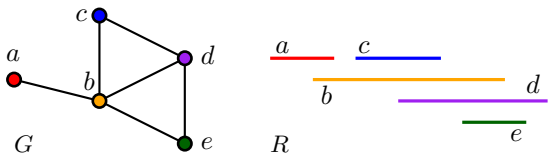



**Figure 1** An interval graph $G$, and an interval representation $R$ of $G$. A valid clique ordering is $\{a, b\}, \{c, b, d\}, \{b, d, e\}$.


31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 90; pp. 90:1–90:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to $\mathcal{R} = (R_1, \ldots, R_k)$ as a *simultaneous representation*. For two graphs $G$, $H$ we define the *intersection* $G \cap H = (V(G) \cap V(H), E(G) \cap E(H))$. A *sunflower* simultaneous graph is a simultaneous graph $\mathcal{G} = (G_1, \ldots, G_k)$ where $G_1, \ldots, G_k$ have pairwise the same intersection, i.e., there is a graph $S$ with $S = G_i \cap G_j$ for each $i \neq j$. We call $S$ the *shared graph* of $\mathcal{G}$.

Simultaneous representations have first been studied in the context of graph embeddings where also shared edges have to be represented by the same arc; see [3] for a survey. The notion of simultaneous representations of general intersection graph classes was introduced by Jampani and Lubiw, who gave an $O(n^3)$-algorithm for recognizing simultaneous sunflower permutation graphs (where $n$ is the number of vertices in $G_1 \cup \cdots \cup G_k$) [17], proved NP-completeness for sunflower chordal graphs (as intersection graphs of subtrees of a tree) [17], and gave an $O(n^2 \log n)$-time algorithm for simultaneous interval graphs with $k = 2$ [15]. They left the case $k > 2$ open. The running times of all these algorithms were subsequently reduced to optimal linear time (assuming each input graph is given separately, thus counting the shared graph $k$ times, and assuming that each input graph belongs to the corresponding class) [4, 19].

Since then, the simultaneous representation problem has also been studied for proper and unit interval graphs [20], circle graphs [8] and permutation graphs [17, 19] where $k$ is part of the input. Bok and Jedličková showed that recognizing simultaneous non-sunflower interval graphs is NP-complete [5] if $k$ is part of the input. Similar results hold for simultaneous proper and unit interval representations [20].

A problem closely related to the (sunflower) simultaneous representation problem is *partial representation extension*, where a representation $R$ of a subgraph $H$ of a single input graph $G$ is given, and the question is, whether $G$ has a representation whose restriction to $H$ coincides with $R$. It has been studied extensively for various graph classes, e.g. for interval graphs [18], circular-arc graphs [10], circle graphs [8, 19], as well as proper and unit interval graphs [18]. Bläsius and Rutter gave a linear-time reduction from the partial interval representation problem to the simultaneous interval representation problem on two graphs [4].

We characterize sunflower simultaneous interval graphs in terms of linear orderings of maximal cliques that satisfy certain consecutivity constrains. This allows us to work with an established data structure called *PQ-tree*, which represents linear orderings satisfying given consecutivity constraints. The algorithms of Jampani and Lubiw [15] and Bläsius and Rutter [4] for recognizing simultaneous interval graphs with $k = 2$ input graphs use a similar characterization and they also use PQ-trees. Jampani and Lubiw iteratively add non-maximal cliques to orders of maximal cliques and associate nodes of distinct PQ-trees to achieve necessary compatibilities. On the other hand, Bläsius and Rutter synchronize a PQ-tree $T$ that describes orderings of maximal cliques of both input graphs with two PQ-trees $T_1$, $T_2$ for the two individual input graphs. To this end, they construct PQ-trees for many nodes of $T$ and a 2-SAT formula which describes dependencies between decisions in these trees. While they establish a more general framework for *simultaneous PQ-orderings*, their approach does not work for more than two input graphs for sunflower simultaneous interval graph recognition.

**Our Result.** We show how to recognize sunflower simultaneous interval graphs in linear time (assuming the input graphs are given separately) even when the number of input graphs is part of the input, thereby answering the open question of Jampani and Lubiw [16]. We note that, similar to Bläsius and Rutter, we use a PQ-tree $T$ that describes orderings of the maximal cliques of the input graphs, synchronize it with PQ-trees for the individual

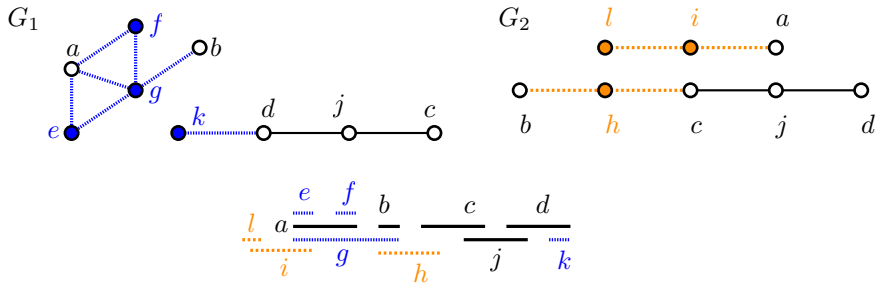



**Figure 2** A simultaneous graph $\mathcal{G} = (G_1, G_2)$ with a simultaneous interval representation of $\mathcal{G}$.

input graphs, and use a 2-SAT formula to describe dependencies between decisions in these PQ-trees. However, with each operation, they only synchronize pairs of PQ-trees while, in a sense, we synchronize multiple PQ-trees at once. Further, we essentially only construct a single PQ-tree for the synchronization, instead of one for potentially each node of $T$, which can be linear in the size of the input. For our construction, we exploit a close relation between consecutivity constraints and certain substructures in PQ-trees (Lemma 5) that provides a converse for a natural and widely used property of PQ-trees and may be of independent interest.

**Organization.** In Section 2, we characterize sunflower interval graphs in terms of linear orderings of maximal cliques and we describe PQ-trees. In Section 3, we describe operations on PQ-trees and dependencies between decisions in the original and resulting PQ-trees. In Section 4, we describe our construction, characterize sunflower interval graphs in terms of this construction, and give the linear-time algorithm for the sunflower interval representation problem. In Section 5 we conclude with open questions.

## 2 Preliminaries

For $n \in \mathbb{N}$ we set $[n] = \{j \in \mathbb{N} \mid 1 \leq j \leq n\}$. In this paper all graphs are simple.

**Simultaneous Interval Graphs.** An *interval representation* $R = \{I_v\}_{v \in V}$ of a graph $G = (V, E)$ associates with each vertex $v \in V$ an interval $I_v = [x, y] \subseteq \mathbb{R}$ such that for each pair of vertices $u, v \in V$ we have $I_u \cap I_v \neq \emptyset \Leftrightarrow uv \in E$; see Figure 1. A *simultaneous interval representation* of a simultaneous graph $\mathcal{G} = (G_1, \ldots, G_k)$ assigns each vertex $v \in \bigcup_{i=1}^{k} V(G_i)$ an interval $I_v$ such that the *induced* interval representation $\{I_v\}_{v \in V(G_i)}$ is an interval representation of $G_i$, for each $i \in [k]$; see Figure 2. In the following we only consider *sunflower simultaneous graphs* $\mathcal{G} = (G_1, \ldots, G_k)$ which are simultaneous graphs for which there is a graph $S$ such that $G_i \cap G_j = S$ for $i \neq j$. Note that it is necessary that $S$ is an induced subgraph of each input graph $G_i$ for $\mathcal{G}$ to be a simultaneous interval graph. We call $S$ the *shared graph* of $\mathcal{G}$.

It is well known that interval graphs can be characterized via orderings of maximal cliques. *A valid clique ordering* of $G$ is a linear ordering of the maximal cliques of $G$ such that for each $v \in V(G)$ the maximal cliques of $G$ that contain $v$ are consecutive.

▶ **Proposition 1** (Fulkerson and Gross [12]). *A graph is an interval graph if and only if it admits a valid clique ordering.*

Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower simultaneous graph with shared graph $S$. For $i \in [k]$, let $\mathcal{K}_i$ denote the set of maximal cliques of $G_i$ and let $\mathcal{K} = \dot{\bigcup}_{i=1}^{k} \mathcal{K}_i$. Note that we use the disjoint union since we want to treat the maximal cliques of the input graphs separately, even if they coincide with maximal cliques of other input graphs. I.e., each clique in $\mathcal{K}$ is tagged with the input graph $G_i$ it comes from. For a vertex $v \in V(G_i)$, we define $\mathcal{K}_i(v) = \{C \in \mathcal{K}_i \mid v \in C\}$ as the set of all maximal cliques of $G_i$ that contain $v$ and for $v \in V(S)$, we define $\mathcal{K}(v) = \{C \in \mathcal{K} \mid v \in C\}$ as the (multi)-set of all maximal cliques of $G_1, \dots, G_k$ that contain $v$. We further define $\mathcal{K}(S)$ as the set of maximal cliques in the shared graph $S$. A *simultaneous clique ordering* of $\mathcal{G}$ is a linear ordering $\sigma$ of $\mathcal{K}$ such that the following two properties hold:

▶ **Property 1.** *For each $v \in V(S)$ the set $\mathcal{K}(v)$ is consecutive.*

▶ **Property 2.** *The restriction of $\sigma$ to $\mathcal{K}_i$ is a valid clique ordering of $G_i$ for each $i \in [k]$.*

The following theorem provides a combinatorial description of sunflower interval graphs which we will use for our algorithm.

▶ **Theorem 2.** *A sunflower simultaneous graph $\mathcal{G}$ is a simultaneous interval graph if and only if it admits a simultaneous clique ordering.*

**Proof.** If $\mathcal{G} = (G_1, \dots, G_k)$ is a simultaneous interval graph, then it has a simultaneous interval representation. For $i \in [k]$, we choose a point $p_C$ for each clique $C \in \mathcal{K}_i$ such that all intervals for $C$ contain $p_C$ and no interval for vertices in $V(G_i) \setminus C$ contains $p_C$. Such a point must exist since intervals have the Helly property and $C$ is maximal in $G_i$. We call these points *clique points*. Then for each $v \in V(S)$ the clique points in the interval $R(v)$ of $v$ are consecutive and exactly the cliques in $\mathcal{K}(v)$. Note that Property 1 does not necessarily hold for non-shared vertices. However, for each input graph $G_i$ the clique points are placed according to the induced representation of $G_i$ and thus provide a clique ordering of $G_i$.

On the other hand, given a simultaneous clique ordering $\sigma$ for $\mathcal{G}$, we construct an interval representation of $\mathcal{G}$ as follows. We first place distinct (clique) points for the maximal cliques in $\mathcal{K}$ on the real line in the order of $\sigma$ from the left to the right. We then set for each vertex $v \in V$ its interval $R(v)$ to $[\ell(v), r(v)]$ where $\ell(v)$ and $r(v)$ are the leftmost point and the rightmost point for cliques containing $v$, respectively. We claim that the clique points and the consecutivity constraints then enforce correct adjacencies. Namely, two vertices $u$, $v$ of the same input graph $G_i$ have intersecting intervals $R(u)$, $R(v)$ if and only if those intervals share a clique point. It remains to show that $R(u)$, $R(v)$ share a clique point if and only if $u, v$ are adjacent. First observe that if $u$, $v$ are adjacent, then there is some maximal clique containing both and its clique point is contained in $R(u)$ and $R(v)$ by Properties 1, 2.

For the other direction, let $R(u), R(v)$ share a clique point. We aim to show that there is a clique containing $u$ and $v$, which implies that $u$ and $v$ are adjacent, concluding the proof. Since $R(u), R(v)$ share a clique point, one of the intervals, lets say $R(u)$, contains an endpoint of the other one. That endpoint $p$ is a clique point for a clique $C_p \in \mathcal{K}_i(v)$ by definition of $R(v)$. If $v \notin V(S)$, then $C_p$ is a clique in $G_i$ that also contains $u$ by Property 2. We can argue analogously if $R(v)$ contains an endpoint of $R(u)$ and $u \notin V(S)$. If both $u$ and $v$ lie in $V(S)$, then $C_p$ contains $u$ and $v$ by Property 1. Finally, consider the case where $u \notin V(S)$, $v \in V(S)$ and $R(v)$ contains no end of $R(u)$, meaning that $R(v) \subseteq R(u)$. Since $S$ has a maximal clique $C_v$ containing $v$ and $G_i$ has a maximal clique $C'_v$ containing $C_v, R(v)$ must contain a clique point for $C'_v$ and thus $u$ and $v$ are contained in clique $C'_v$. ◀

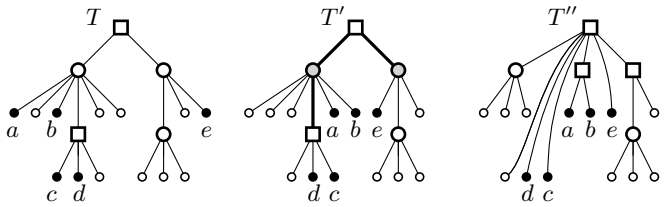



**Figure 3** A PQ-tree $T$, an equivalent PQ-tree $T'$ where the leaves $a, b, c, d, e$ are consecutive, and a reduction $T''$ of $T$ with $\{a, b, c, d, e\}$ (introduced in Section 3). We depict P-nodes as circles and squares as Q-nodes. $T''$ can be obtained from $T'$ by merging the nodes along the thick path after splitting the gray P-nodes. This results in nodes with only two children, which we consider as Q-nodes.

**PQ-Trees.** Let $L$ be a set and let $L' \subseteq L$. The ordering $\leq'$ of $L'$ is *induced* by a linear ordering $\leq$ of $L$, if we have $a \leq' b \Leftrightarrow a \leq b$ for all $a, b \in L'$. A *PQ-tree* is a data structure that represents linear orderings satisfying a set of consecutivity constraints [6]. Formally, a PQ-tree $T$ on a set $L$ of leaves is a rooted ordered tree where each inner node is either a *P-node* or a *Q-node*; see Figure 3. The order of its leaves is the order induced by a preorder traversal of the tree. A PQ-tree $T'$ is *equivalent* to $T$ if $T'$ can be obtained from $T$ by arbitrarily reordering the children of P-nodes and by reversing the order of the children of any subset of Q-nodes. Note that reversing the order of the children of a Q-node $\lambda$ does not change the order of the children of any child of $\lambda$. In this paper, we consider P-nodes with only two children as Q-nodes. Note that this does not affect which PQ-trees are equivalent. For each inner node $\mu$ we say we *flip* $\mu$, if we reverse the order of its children.

A PQ-tree *represents* a linear ordering $\leq$ of $L$ if $\leq$ is the order of the leaves of some equivalent PQ-tree. We write $R(T)$ for all linear orderings of $L$ represented by $T$. The *null-tree* is defined as a special PQ-tree $T_\emptyset$ with $R(T_\emptyset) = \emptyset$. For an inner node $\mu$ of a PQ-tree, let $L(\mu)$ denote the leaves of the subtree rooted at $\mu$. For a leaf $\mu$ let $L(\mu) = \{\mu\}$. We denote the lowest common ancestor of a set $N \subseteq V(T)$ by $\mathrm{lca}_T(N)$. We say that a node $\nu$ of $T$ is left of a node $\lambda$ in $t$ if $\nu$ comes before $\lambda$ in a preorder traversal. However, the order of a node and one of its ancestors will never be relevant in this paper.

## 3 Operations on PQ-Trees

By Theorem 2, we can recognize sunflower simultaneous interval graphs by testing the existence of a simultaneous clique ordering. We use PQ-trees to describe clique orderings that satisfy the properties of a simultaneous clique ordering. Namely, we use a PQ-tree $T$ with leaf set $\mathcal{K}$ to describe all linear orderings satisfying Property 1 and PQ-trees $T'_1, \ldots, T'_k$ where each $T'_i$ represents all valid clique orderings of $G_i$. A simultaneous clique ordering is then a linear ordering $\sigma \in R(T)$ that induces orderings in $R(T'_1), \ldots, R(T'_k)$.

To find such a linear ordering $\sigma$, we "synchronize" these PQ-trees. One step will be to construct a PQ-tree $T_S$ on $\mathcal{K}(S)$ that describes the maximal clique orderings of $S$ that are in some sense compatible with the linear orderings in $R(T'_1), \ldots, R(T'_k)$. Another aspect is to describe the dependencies between decisions at Q-nodes in all constructed PQ-trees.

**Consistency and Backward-Consistency.** We say that an ordered pair of leaves $(\lambda_1, \lambda_2)$ is *forward directed* if $\lambda_1$ comes before $\lambda_2$ in the leaf ordering of the PQ-tree. Otherwise, we call it *backward directed*. Hence, a pair of leaves is either forward directed or backward directed. Note that the corresponding children $\nu_1$, $\nu_2$ of $\mathrm{lca}(\lambda_1, \lambda_2)$ with $\lambda_1 \in L(\nu_1)$ and $\lambda_2 \in L(\nu_2)$

are ordered the same way as $\lambda_1$, $\lambda_2$. Hence, if $\mu$ is a Q-node, flipping $\mu$ changes the order of $\lambda_1$, $\lambda_2$. Let $T_1$, $T_2$ be two PQ-trees on sets $L_1$, $L_2$ with $L_1 \subseteq L_2$ and let $\mu_1$, $\mu_2$ be two Q-nodes in $T_1$, $T_2$ such that there are two leaves $\hat{\lambda}_1$, $\hat{\lambda}_2$ whose order is affected by flipping $\mu_1$ or $\mu_2$, respectively. More formally, let $\mu_1$ be a Q-node in $T_1$ with children $\nu_1^1$, $\nu_2^1$ and let $\mu_2$ be a Q-node in $T_2$ with children $\nu_1^2$, $\nu_2^2$, such that there exist two leaves $\hat{\lambda}_1 \in L(\nu_1^1) \cap L(\nu_1^2)$ and $\hat{\lambda}_2 \in L(\nu_2^1) \cap L(\nu_2^2)$.

We say $\mu_1$ and $\mu_2$ are *consistent*, if for each pair $(\lambda_1, \lambda_2)$ of leaves with $\mu_1 = \text{lca}_{T_1}(\lambda_1, \lambda_2)$, $\mu_2 = \text{lca}_{T_2}(\lambda_1, \lambda_2)$ we have that $(\lambda_1, \lambda_2)$ is forward directed in $T_1$ if and only if it is forward directed in $T_2$. We say $\mu_1$ and $\mu_2$ are *reverse consistent*, if for each pair $(\lambda_1, \lambda_2)$ of leaves with $\mu_1 = \text{lca}_{T_1}(\lambda_1, \lambda_2)$, $\mu_2 = \text{lca}_{T_2}(\lambda_1, \lambda_2)$ we have that $(\lambda_1, \lambda_2)$ is forward directed in $T_1$ if and only if $(\lambda_1, \lambda_2)$ is backward directed in $T_2$. Observe that $\mu_1$, $\mu_2$ cannot be both consistent and reverse consistent at the same time. Consider the case where $\mu_1$, $\mu_2$ are neither consistent nor reverse consistent. Then $\mu_1$, $\mu_2$ order at least one pair of leaves differently (one forward and one backward directed) and at least one pair of leaves the same way (forward or backward directed). This remains true even after flipping one or both of $\mu_1$, $\mu_2$. Hence, in that case no linear order in $R(T_1)$ can be extended to a linear order in $R(T_2)$, since at least one pair of leaves is ordered differently.

We use four operations on PQ-trees for the construction of the PQ-tree $T_S$ that orders the maximal cliques of the shared graph: *reduction*, *intersection*, *projection* and *pruning*. While the first three operations are frequently used for PQ-trees, pruning is less common, but was for example used in the context of level planarity [7] where the algorithm is attributed to Di Battista and Nardelli [2].

To achieve a linear running time for our algorithm, we track what happens to Q-nodes when applying these operations in a PQ-tree, similar as in [4].

**Reduction.** Let $T$ be a PQ-tree with leaf set $L$. The *reduction* of $R(T)$ with a set $L' \subseteq L$ is the set $R'(T, L')$ of linear orderings in $R(T)$ where $L'$ is consecutive. A PQ-tree $T'$ with $R(T') = R'(T, L')$ can be computed from $T$ in $O(|L'|)$ time [6]. This operation is the main operation for PQ-trees, since it allows to compute a PQ-tree $\hat{T}$ on $L$ where sets $S_1, \ldots, S_k \subseteq L$ are consecutive efficiently.

▶ **Proposition 3** (Booth and Lueker [6]). *Let $L$ be a finite set and let $S_1, \ldots, S_k \subseteq L$ be non-empty sets. A PQ-tree $\hat{T}$ on $L$ that represents the linear orderings of $L$ where $S_1, \ldots, S_k$ are consecutive can be computed in $O(|L| + \sum_{i=1}^{k} |S_i|)$ time.*

While the reduction for PQ-trees was originally described by applying a variety of templates, Hsu gave an alternative description [14]; see also [11]. Roughly speaking, we find a certain path $P$ in $T$ that separates $L'$ and $L \setminus L'$, split P-nodes on $P$ suitably and merge the resulting nodes on $P$ to a single P-node. Finally we remove some degeneracies (especially, if $P$ consists of a single P-node $\lambda$, the resulting Q-node is smoothed, effectively just splitting $\lambda$ into two P-nodes); see Figure 3. For more details, we refer to the papers of Booth and Lueker [6] or Hsu [14]. We only use the running time result for the construction of PQ-trees satisfying given consecutivity constraints mentioned above. Especially, we do not need to keep track of the consistencies between Q-nodes for the reduction.

**Intersection.** Let $T_1$, $T_2$ be PQ-trees with leaf set $L$. The *intersection* $T_1 \cap T_2$ is a PQ-tree on $L$ representing $R(T_1) \cap R(T_2)$. It can be computed from $T_1$, $T_2$ in $O(|L|)$ time together with the consistencies between the Q-nodes in $T_1 \cap T_2$ and the Q-nodes in $T_1$ and $T_2$ [19].

For any two cliques $A, B \in L$ where $\text{lca}_{T_1}(A, B)$ or $\text{lca}_{T_2}(A, B)$ is a Q-node, one can see that $\text{lca}_{T_1 \cap T_2}(A, B)$ is a Q-node as follows. Let $\text{lca}_{T_1}(A, B)$ be a Q-node. We can obtain $T_1 \cap T_2$ from $T_1$, by applying a reduction on $T_1$ for each consecutivity constraint of $T_2$. Since

the only possible change to Q-nodes in reductions is being merged with other nodes to a Q-node of higher degree, the lowest common ancestor of $A$, $B$ remains a Q-node (Note that the reduction of a PQ-tree for a given set is unique up to equivalence).

**Projection.** Let $T$ be a PQ-tree with leaf set $L$ and let $L' \subseteq L$. The *projection* $R^\star(T, L')$ from $R(T)$ to $L'$ is the set of linear orderings of $L'$ induced by the linear orderings in $R(T)$ on $L'$. The *projection* $T'$ of $T$ on $L'$ is a PQ-tree on $L'$ with $R(T') = R^\star(T, L')$. It can be computed in $O(|L|)$ time from $T$ by only keeping nodes $\mu$ with $L(\mu) \cap L' \neq \emptyset$ and smoothing all nodes with a single child (that is, removing the node and adding an arc connecting its parent with its child). Here all Q-nodes in $T'$ are *consistent* to their respective copy in $T$.

**Prune.** Let $T$ be a PQ-tree with leaf set $L$, let $\ell' \notin L$ and let $L' \subseteq L$ be consecutive in each $\sigma \in R(T)$. For any $\sigma \in R(T)$, the *prune* of $L'$ to $\ell'$ in $\sigma$ is the result of replacing $L'$ in $\sigma$ by $\ell'$. The *prune* of $L'$ to $\ell'$ in $R(T)$ is the set containing for each $\sigma \in R(T)$ the prune of $L'$ to $\ell'$ in $\sigma$. For pruning PQ-trees, we first observe that consecutive sets of leaves correspond to simple substructures of PQ-trees.

▶ **Lemma 4.** *Let $T$ be a PQ-tree with leaf set $L$ and let $L' \subseteq L$ be consecutive in each $\sigma \in R(T)$. Then, there is either a P-node $\lambda$ with $L(\lambda) = L'$ or a Q-node $\mu$ with a consecutive subset of children $\nu_1, \dots, \nu_l$ such that $\bigcup_{i=1}^{l} L(\nu_i) = L'$.*

**Proof.** We consider $\nu = \mathrm{lca}(L')$, i.e., $L' \subseteq L(\nu)$ and there are two distinct children $\mu_1$,$\mu_2$ of $\nu$ with $L' \cap L(\mu_1) \neq \emptyset$ and $L' \cap L(\mu_2) \neq \emptyset$.

First assume $\nu$ is a P-node. Then for any other child $\xi$ of $\nu$ we have $L(\xi) \subseteq L'$ since otherwise we could violate the consecutivity of $L'$ by placing $\xi$ between $\mu_1$ and $\mu_2$. We further have $L(\mu_1) \subseteq L'$ and $L(\mu_2) \subseteq L'$, since otherwise the consecutivity of $L'$ is violated after flipping $\mu_1$ and $\mu_2$. Hence, we have $L(\nu) = L'$.

Next let $\nu$ be a Q-node, and let $\mu_1, \mu_2$ be the leftmost and the rightmost child of $\nu$ with descendants in $L'$. Since $L'$ is consecutive, all children between $\mu_1$ and $\mu_2$ only have descendants in $L'$. If $\mu_1$ or $\mu_2$ had a descendant not in $L'$, then the consecutivity of $L'$ could be violated by flipping it. Hence, $\nu$ has the stated property. ◀

With this insight, the *prune* of $L'$ to $\ell'$ in $T$ is obtained as follows. By Lemma 4, either $\mathrm{lca}(L')$ is a P-node with $L(\mathrm{lca}(L')) = L'$ or $\mathrm{lca}(L')$ is a Q-node with consecutive children $\nu_1, \dots, \nu_l$ such that $L' = \bigcup_{i=1}^{l} L(\nu_i)$. If $\mathrm{lca}(L')$ is a P-node, the prune is obtained by replacing $\mathrm{lca}(L')$ and its subtree by leaf $\ell'$. Otherwise, the prune is obtained by replacing $\nu_1, \dots, \nu_l$ by $\ell'$ as a child of the Q-node $\mathrm{lca}(L')$. Clearly, given $L'$, the prune can be computed in $O(|L'|)$ time from $T$ with a bottom-up approach. We introduce additional consistencies. Namely, we consider each Q-node $\mu$ of $T$ consistent to its copy $\mu'$ in the prune of $L'$ to $\ell'$ in $T$, if that copy exists. These consistencies are trivial and not explicitly computed.

## 4 Recognition Algorithm

We use Theorem 2 to recognize sunflower simultaneous interval graphs by deciding whether a given sunflower simultaneous graph $\mathcal{G}$ has a simultaneous clique ordering. The rough idea is the following. For Property 1, we construct a PQ-tree $T$ on $\mathcal{K}$ where $\mathcal{K}(v)$ is consecutive for each $v \in V(S)$. For Property 2, we construct for each $i \in [k]$ a PQ-tree $T_i'$ on $\mathcal{K}_i$ where $\mathcal{K}_i(v)$ is consecutive for each $v \in V(G_i)$. I.e., each $T_i'$ represents the valid clique orderings of $G_i$; see Figure 4. By construction, a simultaneous clique ordering of $\mathcal{G}$ is then a linear ordering $\sigma \in R(T)$ that induces a linear ordering in each of $R(T_1'), \dots, R(T_k')$.

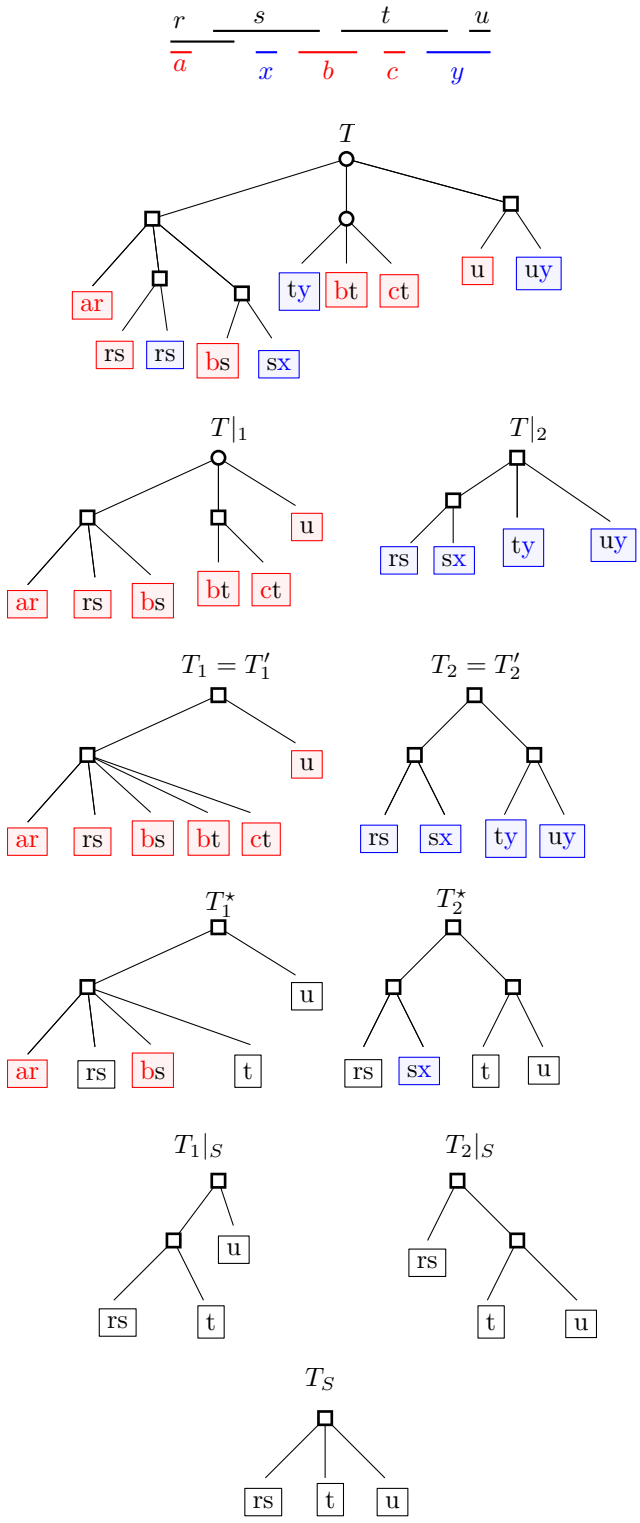



**Figure 4** Top: Simultaneous representation of a sunflower simultaneous graph $\mathcal{G} = (G_1, G_2)$ with $V(G_1) = \{a, b, c, r, s, t, u\}$ and $V(G_2) = \{x, y, r, s, t, u\}$. Below are the PQ-trees for $\mathcal{G}$ with circles for P-nodes and squares for Q-nodes, as defined in Section 4. The leaves are cliques described as strings of the contained vertices.

This means that we obtain $\sigma$ as the order of leaves of $T$, if for $i \in [k]$ any two cliques $A, B \in \mathcal{K}_i$ are ordered the same way in the order of leaves of $T$ and in the order of leaves of $T'_i$. We construct PQ-trees $T_1, \dots, T_k$ by restricting $T'_1, \dots, T'_k$ such that they are "compatible" with $T$ while only loosing linear orderings that do not match any simultaneous clique ordering. We will show that if $\mathrm{lca}_T(A, B)$ is a Q-node, then so is $\mathrm{lca}_{T_i}(A, B)$. This allows to ensure the same ordering of $A$, $B$ in $T$ and $T_i$ by making each pair of backward-consistent Q-nodes consistent. We achieve this with a 2-SAT formula $\Phi$. If $\mu = \mathrm{lca}_T(A, B)$ is a P-node, then we cannot just arrange the children of $\mu$ according to $T_i$, since this can also affect the order of maximal cliques for other input graphs. However, we can resolve this problem as follows. We will see that, if $A$ or $B$ is a child of $\mu$ directly, then its position can be chosen according to $T_i$ without affecting the order of the cliques for other input graphs. Otherwise, $\mu$ has two inner nodes $\nu_1$, $\nu_2$ with $A \in L(\nu_1)$, $B \in L(\nu_2)$ as children. The next lemma then shows that the order of $\nu_1$ and $\nu_2$ is in a sense decided by the order of two shared intervals ("below" $\nu_1$ and $\nu_2$, respectively). This allows us to synchronize $T$ and $T_1, \dots, T_k$ by considering corresponding valid clique orderings for $\mathcal{K}(S)$. Namely, we use the operations on PQ-trees from Section 3 to obtain a PQ-tree $T_S$ on $\mathcal{K}(S)$ that is in a sense compatible with $T_1, \dots, T_k$ and $T$. In $T$, we order the children of a P-node $\mu$ that are inner nodes according to the order of corresponding shared intervals whose order is given by the order of the leaves of $T_S$, and we apply corresponding orderings in each $T_i$. This ensures compatibility of the orders of $T_1, \dots, T_k$, since all children of $\mu$ that are leaves are private to some $T_i$ and can be arranged accordingly in $T$.

Note that each consecutivity constraint for $T$ is a set $\mathcal{K}(v)$ with $v \in V(S)$. In that sense, by the following lemma each child of a P-node in $T$ has a private shared vertex $v \in V(S)$ if it is an inner node.

▶ **Lemma 5.** *Let $L$ be a finite set and let $\{S_1, \dots, S_k\} \subseteq 2^L$ with $|S_i| \geq 2$ for $i \in [k]$. Let $T$ be the PQ-tree on $L$ obtained by making $S_1, \dots, S_k$ consecutive. Let $\mu$ be a P-node and let $\nu$ be a child of $\mu$ that is not a leaf. Then if $\nu$ is a P-node, there is an $S_i$ with $L(\nu) = S_i$. If $\nu$ is a Q-node, there is an $S_i$ with $\bigcup_{j=1}^{l} L(\nu_j) = S_i$ for a consecutive subset of children $\nu_1, \dots, \nu_l$ of $\nu$.*

**Proof.** Let $\nu$ be a P-node and suppose there is no $S_i$ with $L(\nu) = S_i$. First observe that by Lemma 4, for any $S_i$ with $S_i \cap L(\mu) \neq \emptyset$, we have either $L(\mu) \subseteq S_i$ or $S_i \subseteq L(\lambda)$ for some child $\lambda$ of $\mu$. This means that, after contracting the arc $\mu\nu$, no $S_i$ can be violated. However, the contraction allows to order other children of $\mu$ between the children of $\nu$. Thereby $L(\nu)$ is no longer consecutive in all represented linear orderings. This contradicts $T$ originally representing all linear orderings where $S_1, \dots, S_k$ are consecutive, since in each $\sigma \in R(T)$ for the original $T$, leaf set $L(\nu)$ is consecutive.

Next, let $\nu$ be a Q-node and suppose there is no $S_i$ with $\bigcup_{j=1}^{l} L(\nu_j) = S_i$ for any consecutive subset of children $\nu_1, \dots, \nu_l$ of $\nu$. If $\nu$ has precisely two children, we treat it as a P-node and argue as above that there is an $S_i$ with $L(\nu) = S_i$. Hence, assume that $\nu$ has at least three children. By Lemma 4, for any $S_i$ with $S_i \cap L(\mu) \neq \emptyset$, we then have either $L(\mu) \subseteq S_i$ or $S_i \subseteq L(\lambda)$ for some child $\lambda$ of $\mu$. This means that after switching the label of $\mu$ from Q-node to P-node, still all represented linear orderings have all $S_i$ consecutive. This contradicts the choice of $T$, since by making $\mu$ a P-node, $T$ represents additional linear orderings. ◀

Note that Lemma 5 is in a sense the converse of Lemma 4 for the children of P-nodes.

## 4.1 Polynomial-Time Algorithm

We now describe the construction of the 2-SAT formula $\Phi$ and all relevant PQ-trees. For $\Phi$, each Q-node $\lambda$ (of any constructed PQ-tree) is assigned a Boolean variable $x_\lambda$ that tells whether it should be flipped, and we add $(x_\lambda \leftrightarrow x_\mu)$ for consistent nodes $\mu$ and we add $(x_\lambda \not\leftrightarrow x_\nu)$ for backward-consistent nodes $\nu$ to $\Phi$. We describe for which PQ-trees we need to consider the consistencies after introducing all PQ-trees.

Let $\mathcal{G} = (G_1, \dots, G_k)$ be a sunflower simultaneous graph and let $T$ be the PQ-tree on $\mathcal{K}$ that enforces consecutivity of each set $\mathcal{K}(v)$ with $v \in V(S)$; see Figure 4. Note that if $T$ is the null-tree, then there exists no simultaneous clique ordering since Property 1 cannot be satisfied, and by Theorem 2 there is no simultaneous interval representation of $\mathcal{G}$. Hence, we assume in the following, that $T$ is not the null-tree.

For $i \in [k]$, let $T|_i$ be the projection of $T$ on $\mathcal{K}_i$, and let $T_i'$ be the PQ-tree on $\mathcal{K}_i$ that enforces consecutivity of each set $\mathcal{K}_i(v)$ with $v \in V(G_i)$. Note that $T_i'$ describes all valid clique orderings of $G_i$. We are interested in the PQ-tree $T_i = T|_i \cap T_i'$, which restricts the valid clique orderings of $G_i$ to those that are compatible with Property 1; see Figure 4. With Property 2 this means that, if any $T_i$ is the null-tree, then there is no simultaneous clique order. Hence, we assume in the following that no $T_i$ is the null-tree.

We would like to synchronize $T_1, \dots, T_k$ with $T$. However, they have distinct leaf sets. Thus, we cannot just intersect them. Instead, we aim to describe the clique orderings for $S$ that can be induced by $T_1, \dots, T_k$ with PQ-trees. This allows us to find a clique ordering for $S$ that is compatible with all $T_1, \dots, T_k$. With the Q-nodes flipped according to a solution of $\Phi$, this will be enough to synchronize $T_1, \dots, T_k$ with $T$.

We next aim to prune $T_1, \dots, T_k$ to maximal cliques of $S$. For any clique $A \in \mathcal{K}(S)$, we define $\mathcal{K}_i(A)$ as the set of maximal cliques of $G_i$ that contain $A$ as a subclique. It is $\mathcal{K}_i(A) = \{C \in \mathcal{K}_i \mid A \subseteq C\} = \bigcap_{v \in A} \{C \in \mathcal{K}_i \mid v \in C\} = \bigcap_{v \in A} (\mathcal{K}(v) \cap \mathcal{K}_i)$. The critical observation is that, since the intersection of consecutive sets is itself consecutive in a linear order, $\mathcal{K}_i(A)$ is consecutive in $T_i'$ and thus in $T_i$. Note that for distinct $A, B \in \mathcal{K}(S)$, the sets $\mathcal{K}_i(A)$ and $\mathcal{K}_i(B)$ are disjoint, since the set of shared vertices in a maximal clique $C \in \mathcal{K}_i(A) \cap \mathcal{K}_i(B)$ would otherwise be $A$ as well as $B$.

We can now construct for each $T_i$ a PQ-tree describing the corresponding orderings of $\mathcal{K}(S)$ as follows. For $i \in [k]$, let $T_i^\star$ be the PQ-tree obtained by starting with $T_i$ and pruning for each $A \in \mathcal{K}(S)$ the set $\mathcal{K}_i(A)$ to leaf $A$. Then, let $T_i|_S$ be the projection of $T_i^\star$ on $\mathcal{K}(S)$.

Finally, let $T_S = \bigcap_{i=1}^k T_i|_S$ be the intersection of all $T_i|_S$; see Figure 4. By construction, $R(T_S)$ contains all clique orderings of $S$ that can be induced by a simultaneous clique order. Hence, if it is the null-tree, there is no simultaneous clique order.

For each $1 \le i \le k$, we add clauses to $\Phi$ for the consistencies between $T$ and $T|_i$, between $T|_i$ and $T_i$, between $T_i$ and $T_i^\star$, between $T_i^\star$ and $T_i|_S$, and between $T_i|_S$ and $T_S$. If $\Phi$ is not satisfiable, then there is no simultaneous clique ordering. On the other hand, with this, the necessary conditions are also sufficient.

▶ **Theorem 6.** *$(G_1, \dots, G_k)$ is a sunflower interval graph if and only if $\Phi$ is satisfiable and neither $T$ nor $T_S$ is the null-tree.*

**Proof.** If $(G_1, \dots, G_k)$ is a sunflower interval graph the requirements are necessary as discussed above. Hence, assume that neither $T$ nor $T_S$ is the null-tree and that $\Phi$ has a satisfying assignment $\Gamma$. By Theorem 2, it suffices to find a simultaneous clique ordering. We aim to operate on $T$ and each $T_i$ such that the order $\sigma$ of $T$ induces the order of each $T_i$, thus ensuring that $\sigma$ is a simultaneous clique ordering. We first flip all Q-nodes according to $\Gamma$ (that is, flip each Q-node $\lambda$ where $x_\lambda$ is *true*). This ensures that any two cliques $A$, $B$

in $\mathcal{K}$ or $\mathcal{C}(S)$ are ordered the same way in each of $T, T_1, \ldots, T_k$ or $T_1|_S, \ldots, T_k|_S, T_S$ where $\text{lca}(A, B)$ is a Q-node and the sets $\mathcal{K}_i(A)$, $\mathcal{K}_i(B)$ are ordered in $T_i$ the same way as $A$, $B$ are ordered in $T_i|_S$, for $i \in [k]$.

We next order the children of the P-nodes of $T$. Let $\mu$ be a P-node of $T$. Then, by Lemma 5 for each child $\nu$ of $\mu$ that is an inner node, there is a vertex $v \in S$ such that $\mathcal{K}(v) \subseteq L(\nu)$: We choose an arbitrary clique $C_\nu \in \mathcal{K}(S)$ that contains $v$. We then order the children $\nu$ of $\mu$ that are inner nodes according to the order of the corresponding cliques $C_\nu$ given by $T_S$. For $i \in [k]$, we order the sets $\mathcal{K}_i(C_\nu)$ in $T_i$ accordingly. First note that these sets are not empty since each $G_i$ contains a clique containing $C_\nu$. Next note that this orders any two leaves $\lambda_1$,$\lambda_2 \in \mathcal{K}_i$ with $\mu = \text{lca}_T(\lambda_1, \lambda_2)$ that are not children of $\mu$ the same way in $T_i$ as in $T$ since for the corresponding children $\nu_1$, $\nu_2$ of $\mu$ we have that $L(\nu_1)$ and $L(\nu_2)$ are consecutive in $T$ and thus also in $T_i$. E.g., even if $\nu_1$ is a Q-node and $\mathcal{K}_i(C_{\nu_1})$ does not contain $\lambda_1$, this rearrangement still ensures the correct ordering of $\lambda_1$ and $\lambda_2$ in $T_i$. Finally note that this does not flip any Q-nodes of $T_i$ since projection and intersection preserve Q-nodes that order two leaves of the projection set [4]. I.e., for each pair of cliques $A, B \in \mathcal{K}(S)$ such that the order of $\mathcal{K}_i(A)$, $\mathcal{K}_i(B)$ is decided by a Q-node $\mu$ in $T_i$, there is a Q-node in $T_S$ deciding the order of $A$, $B$ the same way as $\mu$ orders $\mathcal{K}_i(A)$, $\mathcal{K}_i(B)$ (after flipping Q-nodes according to $\Gamma$).

With this, for any P-node $\mu$ of $T$ and any two children $\nu_1$, $\nu_2$ of $\mu$ that are inner nodes, each $T_i$ orders any pair of $A \in L(\nu_1) \cap \mathcal{K}_i$ and $B \in L(\nu_2) \cap \mathcal{K}_i$ the same way as $T$. This allows us to order the children of $\mu$ simultaneously according to $T_1, \ldots, T_k$ where each inner node $\nu$ is ordered as any leaf $C \in L(\nu) \cap \mathcal{K}_i$. Note that each child of $\mu$ that is a leaf is contained in a single $T_i$, i.e., it can be placed solely considering the order in $T_i$ (which ensures the correct order with regards to the children of $\mu$ that are inner nodes). Since $L(\nu) \cap \mathcal{K}_i$ is consecutive in $T_i$, the choice of $C(\nu)$ does not matter. I.e., we find an ordering of all children of $\mu$, which is compatible with the orderings given by $T_1, \ldots, T_k$. It remains to show that $T$ now actually provides a simultaneous clique ordering. Property 1 is satisfied by the definition of $T$. For Property 2 we verify that the order of $\mathcal{K}_i$ induced by $T$ is the same as the one given by $T_i$. Consider any two cliques $A, B \in \mathcal{K}_i$. If $\text{lca}_T(A, B)$ is a Q-node, then $A$, $B$ are ordered the same way in $T$ and $T_i$, since we flipped the Q-nodes according to $\Phi$. If $\text{lca}_T(A, B)$ is a P-node, then they are ordered the same way in $T$ and $T_i$ by the operations we just applied on the P-nodes of $T$. ◀

All three requirements in Theorem 6 are necessary; see Figure 5. Theorem 6 allows to recognize sunflower interval graphs in polynomial time by constructing $T, T_S$ and $\Phi$. If $\mathcal{G}$ is a simultaneous interval graph, we obtain a simultaneous clique ordering of $\mathcal{G}$ by following the construction in the proof of Theorem 6. With the construction in Theorem 2, we then obtain a simultaneous interval representation.

▶ **Corollary 7.** *Sunflower interval graphs can be recognized in polynomial time. For yes-instances a simultaneous interval representation can also be constructed in polynomial time.*

## 4.2 Linear-Time Algorithm

To achieve a linear running time, we use that the construction steps can be done efficiently, while also computing the consistencies between Q-nodes, as discussed in Section 3. However, we cannot afford to compute the projection from $T$ on $\mathcal{K}_i$ for each $i \in [k]$ separately, since this could result in an additional factor of $k$ for the running time. Instead, we use the next lemma to compute the projections simultaneously. A similar argumentation has been used by Münch et al. [19].

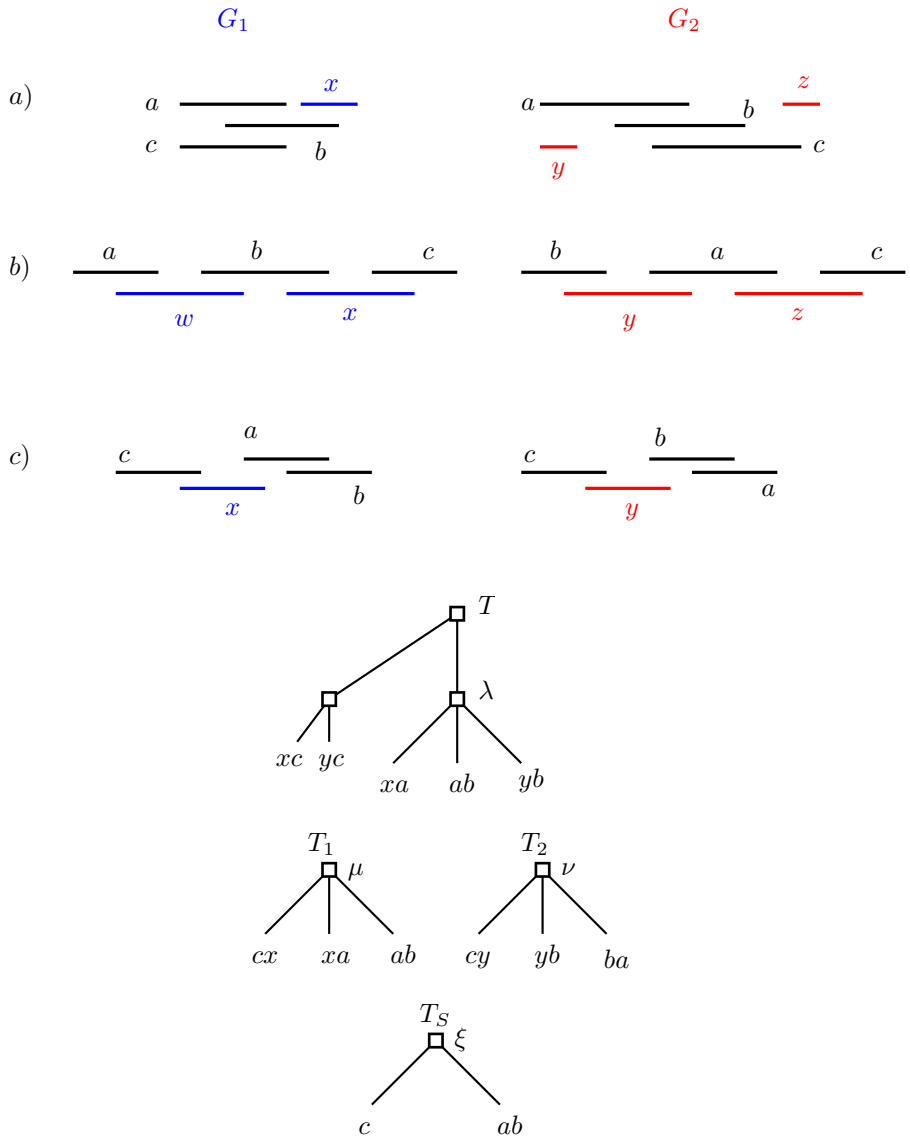



**Figure 5** Variants of a sunflower graph $\mathcal{G} = (G_1, G_2)$ where $G_1$ and $G_2$ are interval graphs, but $\mathcal{G}$ is not a simultaneous interval graph. a) $T$ is a null-tree since the sets $\{abc, bx\}$, $\{abc, ay\}$ and $\{abc, cz\}$ have to be consecutive, while $abc$ can only have two neighbors in the linear ordering. b) $T_S$ is a null-tree since $G_1$ forces $b$ to be in the middle of $a$ and $c$, while $G_2$ forces $a$ to be in the middle of $b$ and $c$. c) $\Phi$ cannot be satisfied since $\xi$ is consistent to $\mu$ and $\nu$ while $\lambda$ is consistent to $\mu$ and backward-consistent to $\nu$ (note that these consistencies are implied in $\Phi$ via the other constructed PQ-trees).

▶ **Lemma 8.** *Let $T$ be an ordered tree with leaf set $L$ and let $\mathcal{S} = \{S_1, \dots, S_l\} \subseteq 2^L$. Then, for each $S_i \in \mathcal{S}$ the projection $T_i^\star$ of $T$ on $S_i$ can be computed such that each node of $T_i^\star$ holds a reference to its original copy in $T$, with a total running time in $O(|L| + \sum_{i=1}^{l} |S_i|)$.*

**Proof.** Observe that a preorder traversal of a $T_i^\star$ is a subsequence of a preorder traversal of $T$. We create a list $U$ that contains a tuple $(S_i, \lambda.p)$ where $\lambda.p$ is the position of $\lambda$ in the preorder of $T$, for each $S_i \in \mathcal{S}$ and $\lambda \in S_i$. We then sort $U$ lexicographically in linear time using radix sort [9]. Then, for each set $S_i \in \mathcal{S}$, the tuples with $S_i$ are consecutive and provide the order of the leaves in $S_i$ in the preorder traversal. For any $S_i$, we find all nodes of $T_i^\star$ in $T$ with the following observation. Let $\mu$ be a node of $T_i^\star$ and let $\nu_1$, $\nu_2$ be two children of $\mu$ with $\nu_1 < \nu_2$ in the preorder. Then $\mu$ is the lowest common ancestor of the rightmost leaf in $L(\nu_1)$ and the leftmost leaf in $L(\nu_2)$. Hence, each inner node of $T_i^\star$ is the lowest common ancestor of two consecutive leaves. We use the lowest common ancestor data structure for static trees by Harel and Tarjan [13], to compute for every pair of nodes $\lambda_1$, $\lambda_2$ in $S_i$ that are consecutive in the preorder the least common ancestor $\mathrm{lca}_T(\lambda_1, \lambda_2)$ and place it between $\lambda_1$ and $\lambda_2$ in a copy $U_i$ of $S_i$ ordered as the preorder. This is already preparing the last step, which is to compute for each node of $T_i^\star$ its parent. Now every node $\lambda \in S_i$ is descendant of both its neighbors $\mu_1$, $\mu_2$ in $U_i$. If $\mu_1$, $\mu_2$ are distinct, the later one in the preorder is the parent of $\lambda$ (and descendant of the other one). By removing $S_i$ and then all duplicates from $U_i$, we get a list of all nodes of $S_i$ of height 1. Note that all duplicates of a node $\lambda$ are consecutive, when they are removed. By repeating the same for each height, we get the parents of all nodes. ◀

With that, the construction of the PQ-trees is straightforward.

▶ **Corollary 9.** *Sunflower interval graphs can be recognized in $O(\sum_{i=1}^{k}(|V(G_i)| + |E(G_i)|))$ time, where $(G_1, \dots, G_k)$ is the input sunflower graph. For yes-instances a simultaneous interval representation can be constructed in the same asymptotic running time.*

**Proof.** We follow the construction of $T$, $T_1, \dots, T_k$, $T_1|_S, \dots, T_k|_S$, $T_S$ for Theorem 6. For each constructed PQ-tree, we maintain the consistencies to the PQ-tree(s) it is constructed from (except for consistencies to unchanged copies of a Q-node, where we use the same variable). The trees $T$ and $T'_1, \dots, T'_k$, can be constructed in $O(\sum_{i=1}^{k}(|V(G_i)| + |E(G_i)|))$ time by Proposition 3. Then, $T|_1, \dots, T|_k$ can be constructed in $O(\sum_{i=1}^{k}(|V(G_i)| + |E(G_i)|))$ time by Lemma 8. The PQ-trees $T_1, \dots, T_k$ can be constructed in the same total time [19]. $T_1|_S, \dots, T_k|_S$ can then be constructed in $O(\sum_{i=1}^{k}(|V(G_i)| + |E(G_i)|))$ by computing the projection and prunes directly. However, we store the smoothed nodes and pruned subtrees (or sets of subtrees), such that we can compute easily a linear order in $R(T_i)$ whose prune is a given linear order in $R(T_i|_S)$ (after projection to $\mathcal{K}(S)$). Finally, $T_S$ can be computed in $O(k \cdot (|V(S)| + |E(S)|))$ time. The 2-SAT formula $\Phi$ can easily be computed from the maintained consistencies. A solution for $\Phi$ can be computed in linear time [1]. With Theorem 6 this suffices to decide whether there is a simultaneous interval representation.

To compute such a representation in linear time, we construct the simultaneous clique ordering a bit differently than in Theorem 6. We first operate on $T_1|_S, \dots, T_k|_S$ such that their leaves are ordered as in $T_S$. This can be done in $O(k \cdot (|V(S)| + |E(S)|))$ time. Then we reverse the smoothing and pruning from $T_1, \dots, T_k$ (using the stored nodes and subtrees) to obtain corresponding linear orderings in $R(T_1), \dots, R(T_k)$. This can be done in time linear in the size of $T_1, \dots, T_k$, i.e., in $O(\sum_{i=1}^{k}(|V(G_i)| + |E(G_i)|))$ time.

In the proof of Theorem 6 we established that the obtained linear orderings can now be merged to a simultaneous clique ordering. We only need to ensure the consecutivities for the shared vertices. Then, we compute for $i \in [k]$ the first and last position $s_i^v$, $t_i^v$ of each

shared vertex $v \in V(S)$ in the clique ordering of $T_i$ by iterating over the clique ordering once. For $i \in [k]$, we sort a list of all these positions in $O(|V(G_i)| + |E(G_i)|)$ time using counting sort [9]. After removing duplicates of positions appearing multiple times, we assign each shared vertex $v$ the positions $\hat{s}_i^v$, $\hat{t}_i^v$ of $s_i^v$, $t_i^v$ in that sorted list. We can then consider for each vertex $v \in V(S)$ two $k$-tuples $\hat{s}^v = (\hat{s}_1^v, \dots, \hat{s}_k^v)$ and $\hat{t}^v = (\hat{t}_1^v, \dots, \hat{t}_k^v)$. We sort a list $L_S$ of all these $k$-tuples lexicographically in $O(k \cdot |V(S)|)$ time using radix sort [9]. This provides us with an order of the start and endpoints of the intervals of the shared vertices in a simultaneous interval representation.

We get a simultaneous clique ordering $\sigma$ by simultaneously iterating over the sorted list $L_S$ and the clique orderings of $T_1, \dots, T_k$ as follows. At each entry $s^v$ (or $t^v$) of $L_S$, append to $\sigma$ for each $T_i$ all cliques between the last appended clique and position $s_i^v$ (or $t_i^v$). After the last entry of $L_S$, append the remaining cliques to $\sigma$. Since we followed the construction of the proof of Theorem 6 there is a simultaneous clique ordering inducing the clique orderings of $T_1, \dots, T_k$. Thus, we have for any two entries $r = (r_1, \dots, r_k)$, $r' = (r'_1, \dots, r'_k)$ that $r \leq r'$ in $L_S$ only if $r_i \leq r'_i$, for all $i \in [k]$. This ensures that each clique $C$ is appended when $C \cap V(S)$ are precisely the shared vertices $v$ for which $s^v$ is processed, but $t^v$ is not. Hence, Property 1 is satisfied and $\sigma$ actually is a simultaneous clique ordering. With that a simultaneous interval representation can be computed straightforwardly, by placing a point for each clique on the real line in that order and then assigning to each vertex $v$ the interval $[s_v, t_v]$ where $s_v$, $t_v$ are the points for the first and last clique containing $v$, as done for Theorem 2. ◀

## 5 Open Questions

While we solve the sunflower representation problem for interval graphs in linear time if each input graph is given separately, a more compact input description is possible, if the number of non-shared vertices is very small. In that case, the input can be given as the union $G = \bigcup_{i=1}^{k} G_i$ as a single graph with labels describing which input graph a non-shared vertex belongs to. Our approach would then have a running time in $O(k \cdot (|V(G)| + |E(G)|))$.

▶ **Question 1.** *Can the sunflower representation problem for interval graphs be solved in $o(k \cdot (|V(G)| + |E(G)|))$ time if the input is given as the union graph $G$?*

Note that it is not clear how to even verify that each input graph is an interval graph with less time.

While the general simultaneous representation problem is NP-complete for interval graphs if the number of input graphs is part of the input, and it is solvable in linear time for $k = 2$, we do not know the complexity for fixed $k > 2$.

▶ **Question 2.** *Can the simultaneous representation problem for interval graphs be solved in polynomial time for a fixed $k > 2$? In particular, considering $k$ as a parameter, is the problem in XP? Is it FPT?*

## References


1 Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

2 Giuseppe Di Battista and Enrico Nardelli. Hierarchies and planarity theory. *IEEE Trans. Syst. Man Cybern.*, 18(6):1035–1046, 1988. doi:10.1109/21.23105.

3 Thomas Bläsius, Stephen G. Kobourov, and Ignaz Rutter. Simultaneous embedding of planar graphs. *CoRR*, abs/1204.5853, 2012. arXiv:1204.5853.

4 Thomas Bläsius and Ignaz Rutter. Simultaneous PQ-ordering with applications to constrained embedding problems. *ACM Trans. Algorithms*, 12(2):16:1–16:46, 2016. doi:10.1145/2738054.

5 Jan Bok and Nikola Jedličková. A note on simultaneous representation problem for interval and circular-arc graphs. *arXiv preprint*, 2018. arXiv:1811.04062.

6 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

7 Guido Brückner and Ignaz Rutter. Partial and constrained level planarity. In Philip N. Klein, editor, *Symposium on Discrete Algorithms, SODA*, pages 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.

8 Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending partial representations of circle graphs. *J. Graph Theory*, 91(4):365–394, 2019. doi:10.1002/jgt.22436.

9 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, Cambridge, 2009.

10 Jirí Fiala, Ignaz Rutter, Peter Stumpf, and Peter Zeman. Extending partial representations of circular-arc graphs. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science – 48th International Workshop, WG 2022*, volume 13453 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2022. doi:10.1007/978-3-031-15914-5_17.

11 Simon D. Fink, Matthias Pfretzschner, and Ignaz Rutter. Experimental comparison of pc-trees and pq-trees. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 43:1–43:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.43.

12 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

13 Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

14 Wen-Lian Hsu. PC-trees vs. PQ-trees. In Jie Wang, editor, *Computing and Combinatorics, 7th Annual International Conference, COCOON*, volume 2108 of *Lecture Notes in Computer Science*, pages 207–217. Springer, 2001. doi:10.1007/3-540-44679-6_23.

15 Krishnam Raju Jampani and Anna Lubiw. Simultaneous interval graphs. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation: 21st International Symposium, ISAAC 2010, Jeju Island, Proceedings, Part I*, pages 206–217. Springer, 2010. doi:10.1007/978-3-642-17517-6_20.

16 Krishnam Raju Jampani and Anna Lubiw. Simultaneous interval graphs. In *Algorithms and Computation*, pages 206–217. Springer, 2010.

17 Krishnam Raju Jampani and Anna Lubiw. The simultaneous representation problem for chordal, comparability and permutation graphs. *Journal of Graph Algorithms and Applications*, 16(2):283–315, 2012. doi:10.7155/jgaa.00259.

18 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomáš Vyskočil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 77(4):1071–1104, April 2017. doi:10.1007/s00453-016-0133-z.

19 Miriam Münch, Ignaz Rutter, and Peter Stumpf. Partial and simultaneous transitive orientations via modular decompositions. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC*, volume 248 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.51.

20 Ignaz Rutter, Darren Strash, Peter Stumpf, and Michael Vollmer. Simultaneous representation of proper and unit interval graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *Proceedings of the 27th Annual European Symposium on Algorithms (ESA'19)*, volume 144 of *LIPIcs*, pages 80:1–80:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.80.

# Relaxed Models for Adversarial Streaming: The Bounded Interruptions Model and the Advice Model

**Menachem Sadigurschi**
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

**Moshe Shechner**
Blavatnik School of Computer Science, Tel Aviv University, Israel

**Uri Stemmer**
Blavatnik School of Computer Science, Tel Aviv University, Israel
Google Research, Tel Aviv, Israel

## Abstract

Streaming algorithms are typically analyzed in the *oblivious* setting, where we assume that the input stream is fixed in advance. Recently, there is a growing interest in designing *adversarially robust* streaming algorithms that must maintain utility even when the input stream is chosen *adaptively and adversarially* as the execution progresses. While several fascinating results are known for the adversarial setting, in general, it comes at a very high cost in terms of the required space. Motivated by this, in this work we set out to explore intermediate models that allow us to interpolate between the oblivious and the adversarial models. Specifically, we put forward the following two models:

- *The bounded interruptions model*, in which we assume that the adversary is only partially adaptive.
- *The advice model*, in which the streaming algorithm may occasionally ask for one bit of advice.

We present both positive and negative results for each of these two models. In particular, we present generic reductions from each of these models to the oblivious model. This allows us to design robust algorithms with significantly improved space complexity compared to what is known in the plain adversarial model.

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

**Keywords and phrases** streaming, adversarial streaming

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.91

**Related Version** *Full Version*: https://arxiv.org/abs/2301.09203

**Funding** This project was partially supported by the Israel Science Foundation (grant 1871/19) and by Len Blavatnik and the Blavatnik Family foundation.

## 1 Introduction

Streaming algorithms are algorithms for processing data streams in which the input is presented as a sequence of items. Generally speaking, these algorithms have access to limited memory, significantly smaller than what is needed to store the entire data stream. This field was formalized by Alon, Matias, and Szegedy [3], and has generated a large body of work that intersects many other fields in computer science. In this work, we focus on streaming algorithms that aim to track a certain function of the input stream, and to continuously report estimates of this function. Formally,

▶ **Definition 1.1** (Informal version of Definition 2.1). *Let $X$ be a finite domain and let $g : X^* \to \mathbb{R}$ be a function that maps every input $\vec{x} \in X^*$ to a real number $g(\vec{x}) \in \mathbb{R}$.*


*Let $\mathcal{A}$ be an algorithm that in every round $i \in [m]$ obtains an element $x_i \in X$ and outputs a response $z_i \in \mathbb{R}$. Algorithm $\mathcal{A}$ is said to be an* oblivious streaming algorithm *for $g$ with accuracy $\alpha$, failure probability $\beta$, and stream length $m$, if the following holds for every input sequence $\vec{x} = (x_1, x_2, \ldots, x_m) \in X^m$. Consider an execution of $\mathcal{A}$ on the input stream $\vec{x}$. Then,* $\Pr[\forall i \in [m]$ *we have* $z_i \in (1 \pm \alpha) \cdot g(x_1, \ldots, x_i)] \geq 1 - \beta$, *where the probability is taken over the coins of algorithm $\mathcal{A}$.*

Note that in Definition 1.1, the streaming algorithm is required to succeed (w.h.p.) for every *fixed* input stream. In particular, it is assumed that the choice for the elements in the stream is *independent* from the internal randomness of the streaming algorithm. This assumption, called the *oblivious setting*, is crucial for the correctness of most classical streaming algorithms. In this work, we are interested in a setting where this assumption does not hold, referred to as the *adversarial setting*.

## 1.1 The (Plain) Adversarial Model

The adversarial streaming model, in various forms, was considered by [20, 12, 13, 1, 2, 15, 8, 7, 16, 24, 6, 4]. The adversarial setting is modeled by a two-player game between a (randomized) `StreamingAlgorithm` and an `Adversary`. At the beginning, we fix a function $g : X^* \to \mathbb{R}$. Throughout the game, the adversary chooses the updates in the stream, and is allowed to *query* the streaming algorithm at $T$ time steps of its choice (referred to as "query times"). Formally, for round $i = 1, 2, \ldots, m$:

1. The `Adversary` chooses an update $x_i \in X$ and a query demand $q_i \in \{0, 1\}$, under the restriction that $\sum_{j=1}^{i} q_j \leq T$.
2. The `StreamingAlgorithm` processes the new update $x_i$. If $q_i = 1$ then, the `StreamingAlgorithm` outputs a response $z_i$, which is given to the `Adversary`.

The goal of the `Adversary` is to make the `StreamingAlgorithm` output an incorrect response $z_i$ at some query time $i$ in the stream. Let $g$ be a function defining a streaming problem, and suppose that there is an oblivious streaming algorithm $\mathcal{A}$ for $g$ that uses space $s$. It is easy to see that $g$ can be solved in the adversarial setting using space $\approx s \cdot T$, by running $T$ copies of $\mathcal{A}$ and using each copy for at most one query. The question is if we can do better. Indeed, Hassidim et al. [16] showed the following result.

▶ **Theorem 1.2** ([16], informal). *If there is an* oblivious *streaming algorithm for a function $g$ that uses space $s$, then there is an* adversarially robust *streaming algorithm for $g$ supporting $T$ queries using space $\approx \sqrt{T} \cdot s$.*

Note that when the number of queries $T$ is large, this construction incurs a large space blowup. One way for coping with this is to assume additional restrictions on the function $g$ or on the input stream. Indeed, starting with Ben-Eliezer et al. [7], most of the positive results on adversarial streaming assumed that the input stream is restricted to have a small *flip-number*, defined as follows.

▶ **Definition 1.3** (Flip number [7]). *The $(\alpha, m)$-flip number of an input stream $\vec{x}$ w.r.t. a function $g$, denoted as $\lambda_{\alpha,m}(\vec{x}, g)$, or simply $\lambda$, is the maximal number of times that the value of $g$ changes (increases or decreases) by a factor of $(1 + \alpha)$ during the stream $\vec{x}$.*

Starting from [7], the prior works of [7, 16, 24, 4] presented generic constructions that transform an oblivious streaming algorithm with space $s$ into an adversarially robust streaming algorithm with space $\approx s \cdot \text{poly}(\lambda)$. That is, under the assumption that the flip-number is bounded, these prior works can even support $T = m$ queries. This is useful since the

parameter $\lambda$ is known to be small for many interesting streaming problems in the *insertion-only* model (where there are no deletions in the stream). However, in general it can be as big as $\Theta(m)$, in which case the transformations of [7, 16, 24, 4] come at a very high cost in terms of space. To summarize this discussion, current reductions from the adversarial to the oblivious setting are useful either when the number of queries $T$ is small, or when the flip-number is small.

## 1.2 Our Results

One criticism of the adversarial model is that it is (perhaps) too pessimistic. Indeed, there could be many scenarios that do not fall into the oblivious model, but are still quite far from an "adversarial" setting. Motivated by this, in this work, we set out to explore intermediate models that allow us to interpolate between the oblivious model and the adversarial model. Specifically, we study two such models, which we call the *bounded interruptions model* and the *advice model*.

### 1.2.1 Adversarial Streaming with Bounded Interruptions (ASBI)

Recall that in the plain adversarial model, the adversary is fully adaptive in the sense that it does not commit to the $i$th update before time $i$. We consider a refinement of this setting in which the adversary is only *partially* adaptive. The game begins with the adversary specifying a complete input stream. Throughout the execution, the adversary (who sees the outputs of the streaming algorithm) can adaptively decide to *interrupt* and to replace the suffix of the stream (which has not yet been processed by the streaming algorithm). Formally, we define the following *ASBI game* between a `StreamingAlgorithm` and an `Adversary`.

**Parameters:** $m$ denotes the length of the stream, $T$ denotes the number of allowed queries, and $R$ denotes the number of allowed interruptions.

1. The `Adversary` chooses a stream $\vec{x} = (x_1, x_2, \ldots, x_m) \in X^m$.
2. For round $i = 1, 2, \ldots, m$
   a. The `Adversary` chooses a query demand $q_i \in \{0, 1\}$ and an interruption demand $d_i \in \{0, 1\}$, under the restriction that $\sum_{j=1}^{i} q_j \leq T$ and $\sum_{j=1}^{i} d_j \leq R$.
   b. If $d_i = 1$ then the adversary outputs a new stream suffix $(x'_i, \ldots, x'_m)$ and we override $(x_i, \ldots, x_m) \leftarrow (x'_i, \ldots, x'_m)$.
   c. The `StreamingAlgorithm` obtains the update $x_i$. If $q_i = 1$ then, the `StreamingAlgorithm` outputs a response $z_i$, which is given to the `Adversary`.

That is, the adversary sees all of the outputs given by the streaming algorithm (in query times), and adaptively decides on $R$ places in which it "interrupts" and arbitrarily modifies the rest of the stream. Importantly, the streaming algorithm "does not know" when interruptions occur. The ASBI model limits the adaptivity of the adversary in a natural way, capturing settings in which the number of "adaptivity rounds" can be bounded. It gives us an intuitive interpolation between the oblivious setting (in which $R = 0$) and the full adversarial setting (obtained by setting $R = T$).[1]

We show the following generic result.

[1] As described, the plain adversarial setting is obtained by setting $R = m$. However, an easy argument shows that setting $R > T$ does not give more power to the adversary over the case of $R = T$.

▶ **Theorem 1.4** (informal version of Theorem 3.1)**.** *If there exists an oblivious streaming algorithm for a function $g : X^* \to \mathbb{R}$ using space $s$ then for every $1 \leq R \leq T$ there exists an adversarially robust streaming algorithm for $g$ in the ASBI model that answers $T$ queries and resists $R$ interruptions using space $\approx \min\left\{R, \sqrt{T}\right\} \cdot s$.*

To obtain this result, we build on the *sketch switching* technique introduced by Ben-Eliezer et al. [7]. Intuitively, we maintain $2R$ copies of an oblivious streaming algorithm $\mathcal{A}$, where in every given moment exactly two of these copies are designated as "active". As long as the two active copies produce (roughly) the same estimates, they remain as the "active" copies, and we use their estimates as our response. Once they disagree, we discard them both (never to be used again) and designate two (fresh) copies as "active". In Section 3 we show that this construction can be formalized to obtain Theorem 1.4.

▶ **Remark 1.5.** *While our method resembles the sketch switching technique of [7], their technique is tailored to the setting where the flip-number is small (i.e., the value of the target function does not "jump" too many times throughout the stream), and is not directly applicable to the ASBI model. Specifically, in [7] there is only* one *active copy, which is "swithched" the moment its response differs significantly from* the previously released estimate*. This is useful under the assumption that the flip-number is small, because the flip-number bounds the number of switches. However, in the ASBI model we do not assume anything about the flip-number of the stream, and the value of the target function can be completely different at different query times.*

For example, the following is a direct application of Theorem 1.4 for $F_2$ estimation (i.e., estimating the second moment of the frequency vector of the input stream).

▶ **Theorem 1.6** ($F_2$ estimation in the ASBI model, informal)**.** *Let $1 \leq R \leq T$. There exists an adversarially robust $F_2$ estimation algorithm in the ASBI model that guarantees $\alpha$-accuracy (w.h.p.) for $T$ queries while resisting $R$ interruptions using space $\approx \min\left\{R, \sqrt{T}\right\}/\alpha^2$.*

This improves the state-of-the-art for turnstile streams in the plain adversarial model (from [16]) whenever $R \lesssim \sqrt{T}$.

**A negative result for the ASBI model.** Note that the space blowup of our construction from Theorem 1.4 grows linearly with the number of interruptions $R$. Recall that in the full adversarial model (where $R = T$ for $T$ queries) it is known that a space blowup of $\sqrt{T}$ suffices (see Theorem 1.2). Thus, one might guess that the correct dependence in $R$ in the ASBI model should be $\sqrt{R}$. However, we show that this is generally not the case. Specifically, we show that there exists a streaming problem that can easily be solved in the oblivious setting with small space, but necessitates space linear in $R$ in the ASBI model, provided that the number of queries is large enough (quadratic in $R$). This shows that our upper bound stated in Theorem 1.4 is (nearly) tight in general. Formally,

▶ **Theorem 1.7** (informal version of Corollary 3.8)**.** *For every $R \leq T$ there exists a streaming problem that requires at least $\Omega\left(\min\left\{R, \sqrt{T}\right\}\right)$ space to be solved in the ASBI model with $R$ interruptions and $T$ queries, but can be solved in the oblivious setting using space* $\mathrm{polylog}(T)$.

### 1.2.2 Adversarial Streaming with Advice (ASA)

We put forward another intermediate model, in which the streaming algorithm may occasionally ask for one bit of *advice* throughout the execution. Formally, we consider the following game, referred to as the ASA game, between the `StreamingAlgorithm` and an `Adversary`.

**Parameters:** $m$ denotes the length of the stream, $T \leq m$ denotes the number of allowed queries, and $\eta \in \mathbb{N}$ is a parameter controlling the query/advice rate.

For round $i = 1, 2, \ldots, m$ :

1. The `Adversary` chooses an update $x_i \in X$ and a query demand $q_i \in \{0, 1\}$, under the restriction that $\sum_{j=1}^{i} q_j \leq T$.
2. The `StreamingAlgorithm` processes the new update $x_i$.
3. If $q_i = 1$ then
   a. The `StreamingAlgorithm` outputs a response $z_i$, which is given to the `Adversary`.
   b. If $\left(\sum_{j=1}^{i} q_j\right) \bmod \eta = 0$ then the `StreamingAlgorithm` specifies a predicate $p_i : X^* \to \{0, 1\}$, and obtains $p_i(x_1, x_2, \ldots, x_i)$.

That is, in the ASA model the adversary is allowed a total of $T$ queries, and once every $\eta$ queries the streaming algorithm is allowed to obtain one bit of advice, computed as a predicate of the items in the stream so far.

Unlike our ASBI model, which (we believe) has a strong algorithmic motivation, the main motivation to study the ASA model is theoretical. It gives us an intuitive way to measure the amount of *additional information* that the streaming algorithm needs in order to maintain utility in the adversarial setting.

▶ **Remark 1.8.** *Even though the main motivation for the ASA model is theoretical, it could also be interesting from an algorithmic standpoint in the following context. Consider a streaming setting in which the input stream is fed into both a (low space) streaming algorithm $\mathcal{A}$ and to a server $\mathcal{S}$. The server has large space and can store all the input stream (and, therefore, can in principle solve the streaming problem itself). However, suppose that the server has some communication bottleneck and is busy serving many other tasks in parallel. Hence we would like to delegate as much of the communication as possible to the "cheap" (low space) streaming algorithm $\mathcal{A}$. The ASA model allows for such a delegation, in the sense that the streaming algorithm handles most of the queries itself, and only once every $\eta$ queries it asks for one bit of advice from the server.*

We show the following generic result.

▶ **Theorem 1.9** (informal version of Theorem 4.2)**.** *If there exists an oblivious* linear *streaming algorithm for a function $g : X^* \to \mathbb{R}$ with space $s$, then for every $\eta \in \mathbb{N}$ there exists an adversarially robust streaming algorithm for $g$ in the ASA model with query/advice rate $\eta$ using space $\approx \eta \cdot s^2$.*

To obtain this result, we rely on a technique introduced by Hassidim et al. [16] which uses *differential privacy* [11] to protect not the input data, but rather the internal randomness of the streaming algorithm. Intuitively, this allows us to make sure that the "robustness" of our algorithm deteriorates *slower* than the advice rate, which allows us to obtain an advice-robustness tradeoff.

Note that the space complexity of the algorithm from Theorem 1.9 does not depend polynomially on the number of queries $T$. For example, the following is a direct application of this theorem in the context of $F_2$ estimation.

▶ **Theorem 1.10** ($F_2$ estimation in the ASA model, informal)**.** *Let $\eta \in \mathbb{N}$. There exists an adversarially robust $F_2$ estimation algorithm in the ASA model with query/advice rate $\eta$ that guarantees $\alpha$-accuracy (w.h.p.) using space $\tilde{O}\left(\eta/\alpha^4\right)$.*

▶ **Remark 1.11.** *We stress that there is a formal sense in which the ASA model is "between" the oblivious and the (plain) adversarial models. Clearly, the ASA model is easier than the plain adversarial model, as we can simply ignore the advice bits. On the other hand, a simple argument shows that the ASA model (with any $\eta > 1$) is qualitatively harder than the oblivious setting. To see this, let $\mathcal{A}$ be an algorithm in the ASA model for a function $g$ with query/advice rate $\eta > 1$. Then $\mathcal{A}$ can be transformed into the following oblivious algorithm $\mathcal{A}_{\text{oblivious}}$ for $g$ (that returns an estimate in every time step without getting any advice):*

**An oblivious algorithm $\mathcal{A}_{\text{oblivious}}$**

1. *Instantiate algorithm $\mathcal{A}$ (which expects to operate in the ASA model).*
2. *In every time $i \in [m]$:*
   a. *Obtain an update $x_i \in X$.*
   b. *Duplicate $\mathcal{A}$ (with its internal state) into a shadow copy $\mathcal{A}^{\text{shadow}}$.*
   c. *Feed the update $(x_i, 0)$ to $\mathcal{A}$ and the update $(x_i, 1)$ to $\mathcal{A}^{\text{shadow}}$, and obtain an answer $z_i$ from the shadow copy. Here we use the notation that $(x, q)$ means an update $x$ with a query demand $q$. Note that we only query the shadow copy.*
   d. *Output $z_i$ and erase the shadow copy from memory.*

*As we "rewind" $\mathcal{A}$ after every query, it is never expected to issue an advice-request and so $\mathcal{A}_{\text{oblivious}}$ never issues an advice-request as well. Furthermore, a simple argument shows that $\mathcal{A}_{\text{oblivious}}$ maintains utility in the oblivious setting.*[2]

▶ **Remark 1.12.** *Our construction has the benefit that the predicates specified throughout the interaction are "simple" in the sense that every single one of them can be computed in a streaming fashion. That is, given the predicate $p_i$, the bit $p_i(x_1, x_2, \ldots, x_i)$ can be computed using small space with one pass over $x_1, x_2, \ldots, x_i$.*

**A negative result for the ASA model.** Theorem 1.9 shows a strong positive result in the ASA model for streaming problems that are defined by real valued functions, presenting space complexity that does not depend directly on the number of queries $T$. We compliment this result with a negative result for a simple streaming problem which is *not* defined by a real valued function. Specifically, we consider (a variant of) the well-studied $\ell_0$-sampling problem, where the streaming algorithm must return an (almost) uniformly random element from the set of non-deleted elements. It is known that the $\ell_0$-sampling problem is easy in the oblivious setting (see e.g. [17]) and hard in the plain adversarial setting (see e.g. [1]). Using a simple counting argument, we show that the $\ell_0$-sampling problem remains hard also in the ASA model *even if the query/advice rate is 1*, i.e., even if the streaming algorithm gets an advice bit for *every* query. Formally,

▶ **Theorem 1.13** (informal version of Theorem 4.5)**.** *Every algorithm for solving the $\ell_0$-sampling problem in the ASA model with $T$ queries must use space $\Omega(T)$. Furthermore, this holds even when $\eta = 1$, that is, even if the algorithm gets an advice bit after* every *query.*

[2] To see this, fix an input stream $\vec{x} = (x_1, x_2, \ldots, x_m)$, and fix $j \in [m]$. Note that the distribution of the output given by $\mathcal{A}_{\text{oblivious}}$ in time $j$ when running on $\vec{x}$ is identical to the outcome distribution of $\mathcal{A}$ when running on the stream $((x_1, 0), \ldots, (x_{j-1}, 0), (x_j, 1))$, which must be accurate w.h.p. by the utility guarantees of $\mathcal{A}$ (since there is only 1 query in this alternative stream, then $\mathcal{A}$ gets no advice when running on it). The claim now follows by a union bound over the query times.

## 1.3 Additional Related Works

The adversarial streaming model (in a setting similar to ours) dates back to at least [1], who studied it implicitly and showed an impossibility result for robust $\ell_0$ sampling in sublinear memory. The adversarial streaming model was then formalized explicitly by [15], who showed strong impossibility results for linear sketches. A recent line of work, starting with [7] and continuing with [16, 24, 4, 6] showed *positive* results (i.e., *robust algorithms*) for many problems of interest, under the assumption that the *flip-number* of the stream is bounded. On the negative side, [7] also presented an attack with $O(n)$ number of adaptive rounds on a variant of the AMS sketch, where $n$ is the size of the domain. Later, [19] presented a streaming problem that can be solved in the oblivious setting with polylogarithmic space, but requires polynomial space in the adversarial setting. Thus, the result of [19] separates the oblivious model from the (plain) adversarial model. More recently, [9] and [10] presented attacks on a concrete algorithm, namely `CountSketch`, with a number of queries that is *linear* in the space of the algorithm and while using only a single round of adaptivity. Following [16], the idea of using differential privacy to protect the internal randomness of interactive algorithms was used in additional settings, for example in the context of dynamic algorithms [5] and learning with experts [23].

# 2 Preliminaries

In this work we consider streaming problems which are defined by a real valued function (where the goal is to approximate the value of this function) as well as streaming problems that define a set of valid solutions and the goal is to return one of the valid solutions. The following definition unifies these two objectives for the oblivious setting.

▶ **Definition 2.1** (Oblivious streaming). *Let $X$ be a finite domain and let $g : X^* \to 2^W$ be a function that maps every input $\vec{x} \in X^*$ to a subset $g(\vec{x}) \subseteq W$ of valid solutions (from some range $W$).*

*Let $\mathcal{A}$ be an algorithm that, for $m$ rounds, obtains an element $x_i \in X$ and outputs a response $z_i \in W$. Algorithm $\mathcal{A}$ is said to be an* oblivious streaming algorithm *for $g$ with failure probability $\beta$, and stream length $m$, if the following holds for every input sequence $\vec{x} = (x_1, x_2, \ldots, x_m) \in X^m$. Consider an execution of $\mathcal{A}$ on the input stream $\vec{x}$. Then,*

$$\Pr\left[\forall i \in [m] \text{ we have } z_i \in g(x_1, \ldots, x_i)\right] \geq 1 - \beta,$$

*where the probability is taken over the coins of algorithm $\mathcal{A}$.*

For example, in the problem of estimating the number of distinct elements in the stream, the function $g$ in the above definition returns the interval $g(x_1, \ldots, x_i) = (1 \pm \alpha) \cdot |\{x_1, \ldots, x_i\}|$, where $\alpha$ is the desired approximation parameter.

# 3 Adversarial Streaming with Bounded Interruptions (ASBI)

In this section we present our results for the ASBI model, defined in Section 1.2.1. We begin with our generic transformation.

## 3.1 A Generic Construction for the ASBI Model

We present a generic construction whose space blowup depends only on the number of interruptions $R$ (and not on the number of queries $T$). We therefore assume in our construction that $T = m$, i.e., that the adversary queries the streaming algorithm on *every* time step. Our construction is given in algorithm `RobustInterruptions`. The following theorem specifies its properties.

**Algorithm 1** `RobustInterruptions`.

**Input:** Parameter $R \in \mathbb{N}^+$ bounding the number of possible interruptions.

**Algorithm used:** An oblivious streaming algorithm $\mathcal{A}$ with space $s$, accuracy $\alpha$, and confidence $\beta$.

1. Initialize $2R$ independent instances of algorithm $\mathcal{A}$, denoted as $\mathcal{A}_1^{\text{answer}}, \ldots, \mathcal{A}_R^{\text{answer}}$ and $\mathcal{A}_1^{\text{check}}, \ldots, \mathcal{A}_R^{\text{check}}$. Set $r = 1$.
2. For $i = 1, 2, \ldots, m$:
   a. Obtain the next item in the stream $x_i \in X$.
   b. Feed $x_i$ to *all* of the copies of algorithm $\mathcal{A}$.
   c. Let $z_{r,i}^{\text{answer}}$ and $z_{r,i}^{\text{check}}$ denote the answers returned by $\mathcal{A}_r^{\text{answer}}$ and $\mathcal{A}_r^{\text{check}}$, respectively.
   d. If $z_{r,i}^{\text{answer}} \in (1 \pm 2\alpha) \cdot z_{r,i}^{\text{check}}$ then output $z_{r,i}^{\text{answer}}$. Otherwise, output $z_{r,i}^{\text{check}}$ and set $r \leftarrow r + 1$.
   e. If $r > R$ then FAIL. Otherwise continue to the next iteration.

▶ **Theorem 3.1.** *Fix any function $g$ and fix $\alpha, \beta > 0$. Let $\mathcal{A}$ be an oblivious streaming algorithm for $g$ that uses space $s$ and guarantees accuracy $\alpha$ with failure probability $\beta$. Then for all $R \in \mathbb{N}^+$ there exists an adversarially robust streaming algorithm for $g$ that resists $R$ interruptions and guarantees accuracy $5\alpha$ with failure probability $O(R\beta)$ using space $O(Rs)$.*

▶ **Remark 3.2.** *Recall that Hassidim et al. [16] showed a generic transformation from an oblivious algorithm for a function $g : X^* \to \mathbb{R}$ with space $s$ to an adversarial algorithm (in the plain model) for $g$, answering $T$ queries with space $\approx \sqrt{T} \cdot s$ (see Theorem 1.2). Therefore, combined with the above theorem, we get that for every such function $g$ and every $1 \leq R \leq T$ there exists an adversarially robust streaming algorithm in the ASBI model that answers $T$ queries and resists $R$ interruptions using space $\approx \min\left\{R, \sqrt{T}\right\} \cdot s$.*

Fix an adversary $\mathcal{B}$ and consider the interaction between algorithm `RobustInterruptions` and the adversary $\mathcal{B}$. For $r \in [R]$, let $i_r$ denote the time step in which $z_{r,i_r}^{\text{check}}$ is returned.

▶ **Lemma 3.3.** *Fix $r^* \in [R]$. With probability at least $1 - \beta$, the answers returned by $\mathcal{A}_{r^*}^{\text{check}}$ in times $1, 2, \ldots, i_{r^*}$ are $\alpha$-accurate. That is, for every $1 \leq i \leq i_{r^*}$ it holds that $z_{r^*,i}^{\text{check}} \in (1 \pm \alpha) \cdot g(x_1, \ldots, x_i)$.*

**Proof.** For simplicity, we assume that the adversary $\mathcal{B}$ is deterministic (this is without loss of generality by a simple averaging argument). Fix the randomness of all copies of algorithm $\mathcal{A}$, except for $\mathcal{A}_{r^*}^{\text{check}}$. Let $\texttt{RI}_{r^*}$ be a variant of algorithm `RobustInterruptions` which is identical to `RobustInterruptions` until the time step $i^*$ in which $r$ becomes $r^*$. In times $i \geq i^*$, algorithm $\texttt{RI}_{r^*}$ simply outputs $z_{r^*,i}^{\text{answer}}$, i.e., the answer given by $\mathcal{A}_{r^*}^{\text{answer}}$. Note that $\mathcal{A}_{r^*}^{\text{check}}$ does not exist in algorithm $RI_{r^*}$.

As we fixed the coins of the copies of $\mathcal{A} \neq \mathcal{A}_{r^*}^{\text{check}}$, the interaction between $\mathcal{B}$ and $\texttt{RI}_{r^*}$ is deterministic. In particular, it generates a single stream $\vec{x}_{r^*}$. By the utility guarantees of algorithm $\mathcal{A}_{r^*}^{\text{check}}$, when run on this stream, then with probability at least $1 - \beta$ it maintains $\alpha$-accuracy throughout the stream.

The lemma now follows by observing that until time $i_{r^*}$ the stream generated by the interaction between $\mathcal{B}$ and algorithm `RobustInterruptions` is identical to the stream $\vec{x}_{r^*}$. ◀

▶ **Lemma 3.4.** *With probability at least $1 - R\beta$, all of the answers given by `RobustInterruptions` (before returning FAIL) are $5\alpha$-accurate.*

**Proof.** Follows from a union bound over Lemma 3.3, and by Step 2d of `RobustInterruptions`. ◀

▶ **Lemma 3.5.** *Algorithm* `RobustInterruptions` *returns FAIL with probability at most* $2R\beta$.

**Proof.** Let $j_1, j_2, \ldots, j_R$ denote the time steps in which the adversary conducts interruptions. That is, $j_1$ is the first time in which the adversary switches the suffix of the stream, $j_2$ is the second time this happens, and so on. Also let $p_1, p_2, \ldots, p_R$ denote the time steps in which the parameter $r$ increases during the execution of algorithm `RobustInterruptions`. Specifically, $p_\ell$ is the time $i$ in which $r$ becomes equal to $\ell + 1$. We show that for every $r \in [R]$, with probability at least $1 - 2r\beta$ it holds that $j_r \leq p_r$. (That is, interruptions happen "faster" then $r$ increases.)

The proof is by induction on $r$. For the base case, $r = 1$, let $\vec{x}_1$ denote the first stream chosen by the adversary. By the utility guarantees of $\mathcal{A}$, with probability at least $1 - 2\beta$ we have that both $\mathcal{A}_1^{\text{answer}}$ and $\mathcal{A}_1^{\text{check}}$ are $\alpha$-accurate w.r.t. this stream, in which case $r$ does not increase. Thus, with probability at least $1 - 2\beta$ we have $j_1 \leq p_1$.

The inductive step is similar: Fix $r \in [R]$, and suppose that $j_r \leq p_r$, which happens with probability at least $(1 - 2r\beta)$ by the inductive assumption. Let $\vec{x}_r$ denote the last stream specified by the adversary before time $p_r$. Note that the internal coins of $\mathcal{A}_{r+1}^{\text{answer}}$ and $\mathcal{A}_{r+1}^{\text{check}}$ are independent with this stream. Hence, by the utility guarantees of $\mathcal{A}$, with probability at least $1 - 2\beta$ we have that both $\mathcal{A}_{r+1}^{\text{answer}}$ and $\mathcal{A}_{r+1}^{\text{check}}$ are $\alpha$-accurate w.r.t. this stream, in which case $r$ does not increase. Overall, with probability at least $1 - 2(r+1)\beta$ we have $j_{r+1} \leq p_{r+1}$.

The lemma now follows by recalling that there are at most $R$ interruptions throughout the execution. Hence, with probability at least $1 - 2R\beta$ it holds that $r$ never increases beyond $R$, and the algorithm does not fail. ◀

Theorem 3.1 now follows by combining Lemmas 3.3, 3.4, 3.5.

## 3.2 A Negative Result for the ASBI Model

We show the following negative result.

▶ **Theorem 3.6.** *For every $R$, there exists a streaming problem over domain of size* $\text{poly}(R)$ *and stream length* $\text{poly}(R)$ *that requires at least* $\Omega(R)$ *space to be solved in the ASBI model with $R$ interruptions and $T = R^2$ queries to within constant accuracy (small enough), but can be solved in the oblivious setting using space* $\text{polylog}(R)$.

We next provide an overview for the proof of this theorem (see the full version of this work for more details). At a high level, Theorem 3.6 follows by strengthening the following negative result of Kaplan et al. [19] for the (plain) adversarial model.

▶ **Theorem 3.7** ([19])**.** *For every $T$, there exists a streaming problem over domain of size* $\text{poly}(T)$ *and stream length* $\text{poly}(T)$ *that requires at least* $\sqrt{T}$ *space to be solved in the (plain) adversarial setting with $T$ queries to within constant accuracy (small enough), but can be solved in the oblivious setting using space* $\text{polylog}(T)$.

To obtain their result, Kaplan et al. [19] presented a streaming problem, called the SADA problem, that is easy to solve in the oblivious setting but requires large space to be solved in the adversarial setting. Specifically, they showed a reduction from a hard problem in learning theory (called the *adaptive data analysis (ADA) problem*) to the task of solving the SADA problem in the adversarial setting with small space.

In the ADA problem, the goal is to design a mechanism $\mathcal{A}$ that initially obtains a dataset $D$ consisting of $n$ i.i.d. samples from some unknown distribution $\mathcal{P}$, and then answers $k$ *adaptively chosen queries* w.r.t. $\mathcal{P}$. Importantly, $\mathcal{A}$'s answers must be accurate w.r.t. the underlying distribution $\mathcal{P}$, and not just w.r.t. the empirical dataset $D$. Hardt, Ullman, and Steinke [14, 22] showed that the ADA problem requires a large sample complexity. Specifically, they showed that every efficient[3] mechanism for this problem must have sample complexity $n \geq \Omega(\sqrt{k})$.

Theorem 3.6 follows by the following two observations regarding the negative result of [19] for the SADA problem, and regarding the underlying hardness result of [14, 22] for the ADA problem:

1. In the hardness results of [14, 22] for the ADA problem, the adversary generates the queries using $O(n)$ rounds of adaptivity, where $n$ is the sample size. In more detail, the adversary poses $O(n^2)$ queries throughout the interaction, but these queries are generated in $O(n)$ bulks where queries in the $j$th bulk depend only on answers given to queries of *previous* bulks.
2. The reduction of Kaplan et al. [19] from the ADA problem to the SADA problem maintains the number of adaptivity rounds. That is, the reduction of Kaplan et al. [19] transforms an adversary for the ADA problem that generates the queries in $\ell$ bulks into an adversary for the SADA problem that uses $\ell$ interruptions.

The following is an immediate corollary of Theorem 3.6.

▶ **Corollary 3.8.** *For every $R \leq T$, there exists a streaming problem over domain of size* $\text{poly}(T)$ *and stream length* $\text{poly}(T)$ *that requires at least* $\Omega\left(\min\left\{R, \sqrt{T}\right\}\right)$ *space to be solved in the ASBI model with $R$ interruptions and $T$ queries to within constant accuracy (small enough), but can be solved in the oblivious setting using space* $\text{polylog}(T)$.

Observe that this (nearly) matches our upper bound stated in Theorem 1.4. We remark that Theorem 3.6 and Corollary 3.8 hold even in a model where the streaming algorithm is strengthen and gets an indication during each interruption round.

# 4 Adversarial Streaming with Advice (ASA)

In this section we present our results for the ASA model, defined in Section 1.2.2. We begin with our generic transformation.

## 4.1 A Generic Construction for the ASA Model

Our generic construction for the ASA model transforms an oblivious and *linear* streaming algorithm $\mathcal{A}$ into a robust streaming algorithm in the ASA model. The linearity property that we need is the following. Suppose that three copies of $\mathcal{A}$, call them $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, are instantiated with the *same* internal randomness $r$, and suppose that $\mathcal{A}_1$ processes a stream $\vec{x}_1$ and that $\mathcal{A}_2$ processes a stream $\vec{x}_2$ and that $\mathcal{A}_3$ processes the stream $\vec{x}_1 \circ \vec{x}_2$ (where the operator $\circ$ stands for concatenation). Then there is an operation, denote it as "+", that allows us to obtain an internal state $(\mathcal{A}_1 + \mathcal{A}_2)$ that is identical to the internal state of $\mathcal{A}_3$. Many classical streaming algorithms have this property (for example, the classical AMS sketch for $F_2$ has this property [3]). Formally,

[3] The results of [14, 22] hold for all computationally efficient mechanisms, or alternatively, for a class of unbounded mechanisms which they call *natural* mechanisms.

**Algorithm 2** `RobustAdvice`$(\beta, m, \eta)$.

**Input:** Parameters: $\beta$ is the failure probability, $m$ is the length of input stream and $\eta$ is the advice query cycle.

**Algorithm used:** An oblivious linear streaming algorithm $\mathcal{A}$ with space $s$ for $\alpha$ accuracy.

**Constants calculation:**

1. $k = \Omega(\eta s \log(m/\beta) \log^2(m/(\beta\alpha)))$ is the number of instances of each of the sets "active", "next" and "shadow".
2. $\varepsilon_0 = \frac{\varepsilon}{\sqrt{8\eta k s \ln(1/\delta)}}$ is the privacy parameter of `PrivateMed` executions, where $\varepsilon = 1/100$, $\delta = O(\beta/m)$.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

1. Initialize $k$ independent instances $\mathcal{A}_1^{\text{active}}, \ldots, \mathcal{A}_k^{\text{active}}$ of algorithm $\mathcal{A}$.
2. REPEAT (outer loop)
   a. Initialize $k$ independent instances $\mathcal{A}_1^{\text{next}}, \ldots, \mathcal{A}_k^{\text{next}}$ of algorithm $\mathcal{A}$.
   b. Let $\mathcal{A}_1^{\text{shadow}}, \ldots, \mathcal{A}_k^{\text{shadow}}$ be duplicated copies of $\mathcal{A}_1^{\text{next}}, \ldots, \mathcal{A}_k^{\text{next}}$, where each $\mathcal{A}_i^{\text{shadow}}$ is initiated with the same randomness as $\mathcal{A}_i^{\text{next}}$.
   c. Denote the current time step as $t$. (That is, so far we have seen $t$ updates in the stream.)
   d. REPEAT (inner loop)
      i. Receive next update $x_t$ and a query demand $q_t \in \{0, 1\}$.
      ii. Insert update $x_t$ into each of $\mathcal{A}_1^{\text{active}}, \mathcal{A}_1^{\text{next}}, \ldots, \mathcal{A}_k^{\text{active}}, \mathcal{A}_k^{\text{next}}$.
      iii. If $q_t = 1$ then:
         - Query $\mathcal{A}_1^{\text{active}}, \mathcal{A}_2^{\text{active}}, \ldots, \mathcal{A}_k^{\text{active}}$ and obtain answers $y_{t,1}, y_{t,2}, \ldots, y_{t,k}$
         - Output $z_t \leftarrow$ `PrivateMed`$(y_{t,1}, y_{t,2}, \ldots, y_{t,k})$ with privacy parameter $\varepsilon_0$.
         - If $\left(\sum_{\ell=1}^{t} q_\ell\right) \bmod \eta = 0$ then define the predicate $p_t$ that given a (prefix of a) stream $\vec{x}$ returns the next bit in the inner state of $(\mathcal{A}_1^{\text{shadow}}, \mathcal{A}_2^{\text{shadow}}, \ldots, \mathcal{A}_k^{\text{shadow}})$ after processing the first $t$ updates in $\vec{x}$. Update the corresponding bit in the state of the corresponding $\mathcal{A}_i^{\text{shadow}}$.
         - If $\left(\sum_{\ell=1}^{t} q_\ell\right) \bmod \eta k s = 0$ then EXIT inner loop. Otherwise, CONTINUE inner loop.
   e. For $i \in [k]$ let $\mathcal{A}_i^{\text{active}}.\mathcal{S}_v \leftarrow \mathcal{A}_i^{\text{next}}.\mathcal{S}_v + \mathcal{A}_i^{\text{shadow}}.\mathcal{S}_v$ and let $\mathcal{A}_i^{\text{active}}.\mathcal{S}_R \leftarrow \mathcal{A}_i^{\text{next}}.\mathcal{S}_R$.

▶ **Definition 4.1** (Linear state algorithm). *Let $\mathcal{A}$ be an algorithm with three segments of memory state. The first segment, denoted as $\mathcal{S}_R$, is randomized in the beginning of the execution and then remains fixed. The second segment, denoted as $\mathcal{S}_v$, is an encoding of a vector in $\mathbb{R}^d$. The third segment, denoted as $\mathcal{S}_c$, is the rest of its memory space and is used for other computations. Then, $\mathcal{A}$ is* linear state *w.r.t. its input stream if for any two streams $\vec{x}_1 = (x_1, \ldots, x_l) \in X^l$, $\vec{x}_2 = (x_1, \ldots, x_p) \in X^p$ with lengths of $l, p \in \mathbb{N}$ and three different executions of $\mathcal{A}$ with the same randomized state ($\mathcal{S}_R$) the following holds:*

$$\mathcal{A}(\vec{x}_1 \circ \vec{x}_2).\mathcal{S}_v = \mathcal{A}(\vec{x}_1).\mathcal{S}_v + \mathcal{A}(\vec{x}_2).\mathcal{S}_v$$

*Where $\mathcal{A}(\vec{x}).\mathcal{S}_v$ is the encoded vector $v \in \mathbb{R}^d$ resulting from the input stream $\vec{x}$ encoded in the corresponding memory state.*

Our construction is given in algorithm `RobustAdvice`, and its properties are stated in the following theorem.[4]. The analysis is deferred to the the full version of this work [21].

▶ **Theorem 4.2.** *Fix any real valued function $g$ and fix $\alpha, \beta > 0$ and $\eta \in \mathbb{N}$. Let $\mathcal{A}$ be an oblivious linear-state streaming algorithm for $g$ that uses space $s$ and guarantees accuracy $\alpha$ with failure probability $1/10$. Then there exists an adversarially robust streaming algorithm for $g$ in the ASA model with query/advice rate $\eta$, accuracy $\alpha$, and failure probability $\beta$ using space $O(\eta s^2 \log(m/\beta) \log^2(m/(\beta\alpha)))$.*

## 4.2 A Negative Result for the ASA Model

In this section we show that $\ell_0$-sampling, a classical streaming problem, cannot be solved with sublinear space in the adversarial setting with advice. Consider a turnstile stream $\vec{u} = (u_1, \ldots, u_m)$ where each $u_i = (a_i, \Delta_i) \in [n] \times \{\pm 1\}$. A $\beta$-error $\ell_0$-sampler returns with probability at least $1 - \beta$ an (almost) uniformly random element from

$$\text{support}(u_1, \ldots, u_m) = \left\{ a \in [n] : \sum_{i:a_i=a} \Delta_i \neq 0 \right\},$$

provided that this support is not empty. The next theorem, due to Jowhari et al. [18], shows that $\ell_0$-sampling is easy in the oblivious setting.

▶ **Theorem 4.3** ([18])**.** *There is a streaming algorithm with storage $O\left(\log^2(n) \log(\frac{1}{\beta})\right)$ bits, that with probability at most $\beta$ reports FAIL, with probability at most $1/n^2$ reports an arbitrary answer, and in all other cases produces a uniform sample from* $\text{support}(\vec{u})$.

Nevertheless, as we next show, this is a hard problem in the ASA setting. In fact, our negative result even holds for a simpler variant of the $\ell_0$-sampling problem, in which the algorithm is allowed to return an *arbitrary* element, rather than a random element. Formally,

▶ **Definition 4.4** (The $J_0$ problem)**.** *Let $X$ be a finite domain and let $\mathcal{A}$ be an algorithm that operates on a stream of updates $(u_1, \ldots, u_m) \in (X \times \{\pm 1\})$, given to $\mathcal{A}$ one by one. Algorithm $\mathcal{A}$ solves the $J_0$ problem with failure probability $\beta$ if, except with probability at most $\beta$, whenever $\mathcal{A}$ is queried it outputs an element with non-zero frequency w.r.t. the current stream. That is, if $\mathcal{A}$ is queried in time $i$ then it should output an element from* $\text{support}(u_1, \ldots, u_i)$.

▶ **Theorem 4.5.** *Let $X$ be a finite domain and let $T$ be such that $|X| = \Omega(T)$ (large enough). Let $\mathcal{A}$ be an algorithm for solving the $J_0$ problem over $X$ in the adversarial setting with advice with $T$ queries and with failure probability at most $3/4$. Then $\mathcal{A}$ uses space $\Omega(T)$. Furthermore, this holds also when $\eta = 1$, that is, even if algorithm $\mathcal{A}$ gets an advice after* every *query.*

**Proof.** Let $\mathcal{A}$ be an algorithm for the $J_0$ problem with $T$ queries over domain $X$ in the ASA setting with failure probability at most $3/4$. Consider the following thought experiment.

[4] We assume that the estimates given by the oblivious algorithm $\mathcal{A}$ are in the range $[-m^c, -1/m^c] \cup \{0\} \cup [1/m^c, m^c]$ for some constant $c$, and are rounded to the nearest power of $(1 + \alpha)$. See the full version of this work for more details [21].

**Input:** $Y \subseteq X$ of size $|Y| = T$
1. For every $x \in Y$, feed algorithm $\mathcal{A}$ the update $(x, 1)$.
2. Initiate $\hat{Y} = \emptyset$.
3. Repeat $T$ times:
   a. Query $\mathcal{A}$ and obtain an outcome $x \in X$
   b. If $\mathcal{A}$ requests an advice then give it a random bit $b$.
   c. Add $x$ to $\hat{Y}$
   d. Feed the update $(x, -1)$ to $\mathcal{A}$
4. Output $\hat{Y}$.

We say that the thought experiment *succeeds* if $\hat{Y} = Y$. By the assumption on $\mathcal{A}$, for every input $Y$, our thought experiment succeeds with probability at least $2^{-T}/4$. This is because if all of the bits of advice are correct then $\mathcal{A}$ succeeds with probability at least $1/4$, and the advice bits are all correct with probability at least $2^{-T}$. Hence, there must exist a fixture of $\mathcal{A}$'s coins and a fixture of an advice string $\vec{b}$ for which our thought experiments succeeds on at least $2^{-T}/4$ fraction of the possible inputs $Y$.[5]

That is, after fixing $\mathcal{A}$'s coins and the advice string $\vec{b}$ as above, there is a subset of inputs $\mathfrak{B}$ of size $|\mathfrak{B}| \geq \frac{2^{-T}}{4}\binom{|X|}{T}$ such that for every $Y \in \mathfrak{B}$, when executed on $Y$, our thought experiment outputs $\hat{Y} = Y$. Finally, note that the inner state of algorithm $\mathcal{A}$ at the end of Step 1 *determines* the outcome of our thought experiment. Hence, as there are at least $\frac{2^{-T}}{4}\binom{|X|}{T}$ different outputs, there must be at least $\frac{2^{-T}}{4}\binom{|X|}{T}$ possible different inner states for algorithm $\mathcal{A}$, meaning that its space complexity (in bits) is at least $\log\left(\frac{2^{-T}}{4}\binom{|X|}{T}\right)$, which is more than $T$ provided that $|X| = \Omega(T)$ (large enough). ◀

## References


1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467, 2012. doi:10.1137/1.9781611973099.40.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012. doi:10.1145/2213556.2213560.

3 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

4 Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer. A framework for adversarial streaming via differential privacy and difference estimators. In *ITCS*, volume 251 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

5 Amos Beimel, Haim Kaplan, Yishay Mansour, Kobbi Nissim, Thatchaphol Saranurak, and Uri Stemmer. Dynamic algorithms against an adaptive adversary: generic constructions and lower bounds. In *STOC*, pages 1671–1684. ACM, 2022.

6 Omri Ben-Eliezer, Talya Eden, and Krzysztof Onak. Adversarially robust streaming via dense-sparse trade-offs. In *SOSA@SODA*, pages 214–227, 2022.

7 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *J. ACM*, 69(2):17:1–17:33, 2022.


[5] Otherwise, consider sampling an input $Y$ uniformly. We have that $\frac{2^{-T}}{4} \leq \Pr_{r,\vec{b},Y}[\mathcal{A}_{r,\vec{b}}(Y) \text{ succeeds}] = \sum_{r,\vec{b}} \Pr[r,\vec{b}] \cdot \Pr_Y[\mathcal{A}_{r,\vec{b}}(Y) \text{ succeeds}] < \sum_{r,\vec{b}} \Pr[r,\vec{b}] \cdot \frac{2^{-T}}{4} = \frac{2^{-T}}{4}$, which is a contradiction. Here $r$ denotes the randomness of $\mathcal{A}$ and $\vec{b}$ is the advice string.

8 Omri Ben-Eliezer and Eylon Yogev. The adversarial robustness of sampling. In *PODS*, pages 49–62, 2020.

9 Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, Moshe Shechner, and Uri Stemmer. On the robustness of countsketch to adaptive inputs. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 4112–4140. PMLR, 2022.

10 Edith Cohen, Jelani Nelson, Tamas Sarlos, and Uri Stemmer. Tricking the hashing trick: A tight lower bound on the robustness of countsketch to adaptive inputs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):7235–7243, 2023.

11 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

12 A. C. Gilbert, B. Hemenway, A. Rudra, M. J. Strauss, and M. Wootters. Recovering simple signals. In *Information Theory and Applications Workshop (ITA)*, pages 382–391, 2012.

13 A. C. Gilbert, B. Hemenway, M. J. Strauss, D. P. Woodruff, and M. Wootters. Reusable low-error compressive sampling schemes through privacy. In *IEEE Statistical Signal Processing Workshop (SSP)*, pages 536–539, 2012.

14 Moritz Hardt and Jonathan Ullman. Preventing false discovery in interactive data analysis is hard. In *FOCS*. IEEE, october 19-21 2014.

15 Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *STOC*, pages 121–130, 2013.

16 Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *J. ACM*, 2022. doi:10.1145/3556972.

17 Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *PODS*, pages 49–58. ACM, 2011.

18 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58, 2011.

19 Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming using the bounded storage model. In *CRYPTO*, pages 94–121, 2021.

20 Ilya Mironov, Moni Naor, and Gil Segev. Sketching in adversarial environments. *SIAM J. Comput.*, 40(6):1845–1870, 2011. doi:10.1137/080733772.

21 Menachem Sadigurschi, Moshe Shechner, and Uri Stemmer. Relaxed models for adversarial streaming: The bounded interruptions model and the advice model, 2023. arXiv:2301.09203.

22 Thomas Steinke and Jonathan R. Ullman. Interactive fingerprinting codes and the hardness of preventing false discovery. In *2016 Information Theory and Applications Workshop, ITA 2016, La Jolla, CA, USA, January 31 – February 5, 2016*, pages 1–41. IEEE, 2016. doi:10.1109/ITA.2016.7888199.

23 David P. Woodruff, Fred Zhang, and Samson Zhou. Streaming algorithms for learning with experts: Deterministic versus robust. *CoRR*, abs/2303.01709, 2023. arXiv:2303.01709.

24 David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In *FOCS*, pages 1183–1196, 2022.

# Maximal $k$-Edge-Connected Subgraphs in Almost-Linear Time for Small $k$

**Thatchaphol Saranurak**
University of Michigan, Ann Arbor, MI, USA

**Wuwei Yuan**
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

## Abstract

We give the first almost-linear time algorithm for computing the *maximal $k$-edge-connected subgraphs* of an undirected unweighted graph for any constant $k$. More specifically, given an $n$-vertex $m$-edge graph $G = (V, E)$ and a number $k = \log^{o(1)} n$, we can deterministically compute in $O(m + n^{1+o(1)})$ time the unique vertex partition $\{V_1, \ldots, V_z\}$ such that, for every $i$, $V_i$ induces a $k$-edge-connected subgraph while every superset $V_i' \supset V_i$ does not. Previous algorithms with linear time work only when $k \leq 2$ [Tarjan SICOMP'72], otherwise they all require $\Omega(m + n\sqrt{n})$ time even when $k = 3$ [Chechik et al. SODA'17; Forster et al. SODA'20].

Our algorithm also extends to the decremental graph setting; we can deterministically maintain the maximal $k$-edge-connected subgraphs of a graph undergoing edge deletions in $m^{1+o(1)}$ total update time. Our key idea is a reduction to the dynamic algorithm supporting pairwise $k$-edge-connectivity queries [Jin and Sun FOCS'20].

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis

**Keywords and phrases** Graph algorithms, Maximal $k$-edge-connected subgraphs, Dynamic $k$-edge connectivity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.92

**Related Version** *Full Version*: `https://arxiv.org/abs/2307.00147` [11]

**Funding** *Thatchaphol Saranurak*: Supported by NSF CAREER grant 2238138.

## 1 Introduction

We study the problem of efficiently computing the *maximal $k$-edge-connected subgraphs*. Given an undirected unweighted graph $G = (V, E)$ with $n$ vertices and $m$ edges, we say that $G$ is *$k$-edge-connected* if one needs to delete at least $k$ edges to disconnect $G$. The maximal $k$-edge-connected subgraphs of $G$ is a unique vertex partition $\{V_1, \ldots, V_z\}$ of $V$ such that, for every $i$, the induced subgraph $G[V_i]$ is $k$-edge-connected and there is no strict superset $V_i' \supset V_i$ where $G[V_i']$ is $k$-edge-connected.

This fundamental graph problem has been intensively studied. Since the 70's, Tarjan [13] showed an optimal $O(m)$-time algorithm when $k = 2$. For larger $k$, the folklore recursive mincut algorithm takes $\tilde{O}(mn)$ time[1] and there have been significant efforts from the database community in devising faster heuristics [15, 17, 2, 12, 16] but they all require $\Omega(mn)$ time in the worst case. Eventually in 2017, Chechik et al. [3] broke the $O(mn)$ bound to $\tilde{O}(m\sqrt{n}k^{O(k)})$ using a novel approach based on *local* cut algorithms. Forster et al. [4] then improved the local cut algorithm and gave a faster Monte Carlo randomized algorithm with $\tilde{O}(mk + n^{3/2}k^3)$ running time. Very recently, Geogiadis et al. [5] showed a deterministic algorithm with

[1] The algorithm computes a global minimum cut $(A, B)$ (using e.g. Karger's algorithm [8]) and return $\{V\}$ if the cut size of $(A, B)$ is at least $k$. Otherwise, recurse on both $G[A]$ and $G[B]$ and return the union of the answers of the two recursions.

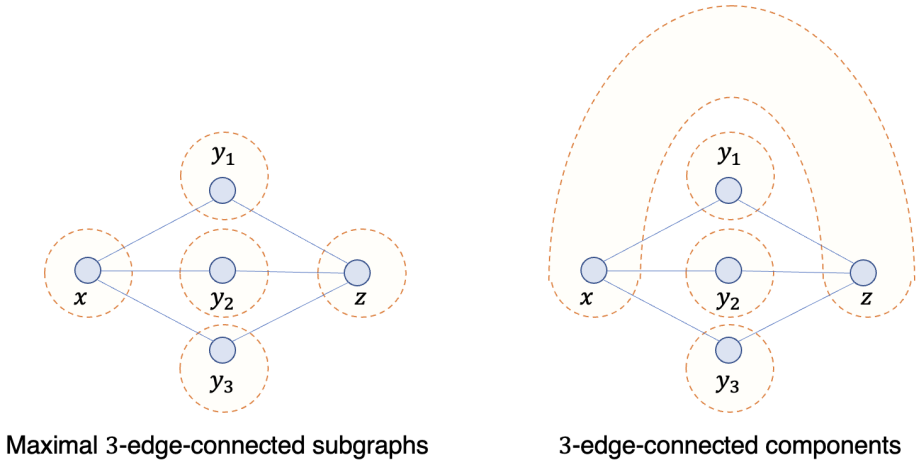




**Figure 1** A graph $G$ where its maximal 3-edge-connected subgraphs are different from its 3-edge-connected components.

$\tilde{O}(m + n^{3/2}k^8)$ time and also how to sparsify a graph to $O(nk \log n)$ edges while preserving maximal $k$-edge-connected subgraphs in $O(m)$ time. Thus, the factor $m$ in the running time of all algorithms can be improved to $O(nk \log n)$ while paying an $O(m)$ additive term. The $O(mn)$ bound has also been improved even in more general settings such as directed graphs and/or vertex connectivity [6, 3, 4] as well as weighted undirected graphs [10]. Nonetheless, in the simplest setting of undirected unweighted graphs where $m = O(n)$ and $k = O(1)$, the $\Omega(n\sqrt{n})$ bound remains the state of the art since 2017.

Let us discuss the closely related problem called *$k$-edge-connected components*. The goal of this problem is to compute the unique vertex partition $\{\hat{V}_1, \ldots, \hat{V}_{z'}\}$ of $V$ such that, each vertex pair $(s, t)$ is in the same part $\hat{V}_i$ iff the $(s, t)$-minimum cut in $G$ (not in $G[\hat{V}_i]$) is at least $k$. The partition of the maximal $k$-edge-connected subgraphs is always a refinement of the $k$-edge-connected components and the refinement can be strict. See Figure 1 for example. Very recently, the Gomory-Hu tree algorithm by Abboud et al. [1] implies that $k$-edge-connected components can be computed in $m^{1+o(1)}$ time in undirected unweighted graphs. This algorithm, however, does not solve nor imply anything to our problem. See Appendix A for a more detailed discussion.

It is an intriguing question whether one can also obtain an almost-linear time algorithm for maximal $k$-edge-connected subgraphs, or there is a separation between these two closely related problems.

## Our results

In this paper, we show the first almost-linear time algorithm when $k = \log^{o(1)} n$, answering the above question affirmatively at least for small $k$.

▶ **Theorem 1.** *There is a deterministic algorithm that, given an undirected unweighted graph $G$ with $n$ vertices and $m$ edges, computes the maximal $k$-edge-connected subgraphs of $G$ in $O(m + n^{1+o(1)})$ time for any $k = \log^{o(1)} n$.*

Our techniques naturally extend to the decremental graph setting.

▶ **Theorem 2.** *There is a deterministic algorithm that, given an undirected unweighted graph $G$ with $n$ vertices and $m$ edges undergoing a sequence of edge deletions, maintains the maximal $k$-edge-connected subgraphs of $G$ in $m^{1+o(1)}$ total update time for any $k = \log^{o(1)} n$.*

Dynamic algorithms for maximal $k$-edge-connected subgraphs were recently studied in [5]. For comparison, their algorithm can handle both edge insertions and deletions but require $O(n\sqrt{n}\log n)$ worst-case update time, which is significantly slower than our $m^{o(1)}$ amortized update time. When $k = 3$, they also gave an algorithm that handles edge insertions only using $\tilde{O}(n^2)$ total update time.

### Previous Approaches and Our Techniques

Our approach diverges significantly from the local-cut-based approach in [3, 4]. In these previous approaches, they call the local cut subroutine $\Omega(n)$ times and each call takes $\Omega(\sqrt{n})$ time. Hence, their running time is at least $\Omega(n\sqrt{n})$ and this seems inherent without significant modification. Recently, [5] took a different approach. Their $\tilde{O}(m + n^{3/2}k^8)$-time algorithm efficiently implements the folklore recursive mincut algorithm by feeding $O(nk)$ updates to the dynamic minimum cut algorithm by Thorup [14]. However, since Thorup's algorithm has $\Omega(\sqrt{n})$ update time, the final running time of [5] is at least $\Omega(n\sqrt{n})$ as well.

Our algorithm is similar to [5] in spirit but is much more efficient. We instead apply the dynamic $k$-edge connectivity algorithms by Jin and Sun [7] that takes only $n^{o(1)}$ update time when $k = \log^{o(1)} n$. Our reduction is more complicated than the reduction in [5] to dynamic minimum cut because the data structure by [7] only supports pairwise $k$-edge connectivity queries, not a global minimum cut. Nonetheless, we show that $\tilde{O}(nk)$ updates and queries to this "weaker" data structure also suffice.

Our approach is quite generic. Our algorithm is carefully designed without the need to check if the graph for which the recursive call is made is $k$-edge-connected. This allows us to extend our algorithm to the dynamic case.

### Organization

We give preliminaries in Section 2. Then, we prove Theorem 1 and Theorem 2 in Section 3 and Section 4, respectively.

## 2 Preliminaries

Let $G = (V, E)$ be an *unweighted undirected* graph. Let $n = |V|$ and $m = |E|$, and assume $m = \text{poly}(n)$ and $k = \log^{o(1)} n$. For any $S, T \subseteq V$, let $E(S, T) = \{(u, v) \in E \mid u \in S, v \in T\}$. For every vertex $u$, the degree of $u$ is $\deg(u) = |\{(u, v) \mid (u, v) \in E\}|$. For every subset of vertices $S \subseteq V$, the volume of $S$ is $\text{vol}(S) = \sum_{u \in S} \deg(u)$. Denote $G[S]$ as the induced graph of $G$ on a subset of vertices $S \subseteq V$.

Two vertices $s$ and $t$ are *$k$-edge-connected* in $G$ if one needs to delete at least $k$ edges to disconnect $s$ and $t$ in $G$. A vertex set $S$ is *$k$-edge-connected* if every pair of vertices in $S$ is $k$-edge-connected. We use the convention that $S$ is $k$-edge-connected when $|S| = 1$. We say that a graph $G = (V, E)$ is $k$-edge-connected if $V$ is $k$-edge-connected. A *$k$-edge-connected component* is an inclusion-maximal vertex set $S$ such that $S$ is $k$-edge-connected. A whole vertex set can always be partitioned into $k$-edge-connected components. We use $kECC(u)$ to denote the unique $k$-edge-connected component containing $u$. Note that a $k$-edge-connected component may not induce a connected graph when $k > 2$. A vertex set $S$ is a *$k$-cut* if $|E(S, V \setminus S)| < k$. Note, however, we also count the whole vertex set $V$ as a trivial $k$-cut.

We will crucially exploit the following dynamic algorithm in our paper.

▶ **Theorem 3** (Dynamic pairwise $k$-edge connectivity [7]). *There is a deterministic algorithm that maintains a graph $G$ with $n$ vertices undergoing edge insertions and deletions using $n^{o(1)}$ update time and, given any vertex pair $(s,t)$, reports whether $s$ and $t$ are $k$-edge-connected in the current graph $G$ in $n^{o(1)}$ time where $k = \log^{o(1)} n$.*

For the maximal $k$-edge-connected subgraph problem, we can assume that the graph is sparse using the *forest decomposition*.

▶ **Definition 4** (Forest decomposition [9]). *A $t$-forest decomposition of a graph $G$ is a collection of forests $F_1, \ldots, F_t$, such that $F_i$ is a spanning forest of $G \setminus \bigcup_{j=1}^{i-1} F_j$, for every $1 \leq i \leq t$.*

▶ **Theorem 5** (Lemma 8.3 of [5]). *Any $O(k \log n)$-forest decomposition of a graph has the same maximal $k$-edge-connected subgraphs as the original graph. Moreover, there is an algorithm for constructing such a $O(k \log n)$-forest decomposition in $O(m)$ time.*

## 3 The Static Algorithm

In this section, we prove our main result, Theorem 1. The key idea is the following reduction:

▶ **Lemma 6.** *Suppose there is a deterministic decremental algorithm supporting pairwise $k$-edge-connectivity that has $t_p \cdot m$ total preprocessing and update time on an initial graph with $n$ vertices and $m$ edges and query time $t_q$.*

*Then, there is a deterministic algorithm for computing the maximal $k$-edge-connected subgraphs in $O(m + (t_p + t_q) \cdot kn \log^2 n)$ time.*

By plugging in Theorem 3, we get Theorem 1. The rest of this section is for proving Lemma 6. Throughout this section, we let $t_q$ denote the query time of the decremental pairwise $k$-edge connectivity data structure that Lemma 6 assumes.

Recall again that, for any vertex $u$, $u$'s $k$-edge-connected component, $kECC(u)$, might not induce a connected graph. The first tool for proving Lemma 6 is a "local" algorithm for finding a connected component of $G[kECC(u)]$.

▶ **Lemma 7.** *Given a graph $G$ and a vertex $u$, there is a deterministic algorithm for finding the connected component $U$ containing $u$ of $G[kECC(u)]$ in $O(t_q \cdot \mathrm{vol}(U))$ time.*

**Proof.** We run BFS from $u$ to explore every vertex in the connected component $U$ containing $u$ of $kECC(u)$. During the BFS process, we only visit the vertices in $kECC(u)$ by checking if the newly found vertex is $k$-edge-connected to $u$. Since each edge incident to $U$ is visited at most twice, the total running time is $O(t_q \cdot \mathrm{vol}(U))$. ◀

Below, we describe the algorithm for Lemma 6 in Algorithm 1 and then give the analysis.

### Correctness

We start with the following structural lemma.

▶ **Lemma 8** (Lemma 5.6 of [3]). *Let $T$ be a $k$-cut in $G[C]$ for some vertex set $C$. Then, either*

- *$T$ is a $k$-cut in $G$ as well, or*
- *$T$ contains an endpoint of $E(C, V(G) \setminus C)$.*

Next, the crucial observation of our algorithm is captured by the following invariant.

**Algorithm 1** MAIN$(G, L)$: compute the maximal $k$-edge-connected subgraphs.

**Input:** An undirected connected graph $G = (V, E)$, and a list of vertices $L$ (initially $L = V$). Note that the parameters are passed by value.
**Output:** The maximal $k$-edge-connected subgraphs of $G$.

```
1  S ← ∅.
2  while |L| > 1 do
3      Choose an arbitrary pair (u, v) ∈ L.
4      if u and v are k-edge-connected in G then
5          L ← L \ {v}.
6      else
7          Simultaneously compute the u's connected component of G[kECC(u)] and
           the v's connected component of G[kECC(v)], until the one with the smaller
           volume (denoted by U) is found.
8          S ← S ∪ Main(G[U], U).
9          G ← G \ U.
10         L ← (L \ U) ∪ {w ∉ U | (x, w) ∈ E(U, V(G) \ U)}.
11     end
12 end
13 S ← S ∪ {V(G)}.
14 return S.
```

▶ **Lemma 9.** *At any step of Algorithm 1, every $k$-cut $T$ in $G$ is such that $T \cap L \neq \emptyset$.*

**Proof.** The base case is trivial because $L \leftarrow V$ initially. Next, we prove the inductive step. $L$ can change in Line 5 or Line 8.

In the first case, the algorithm finds that $u$ and $v$ are $k$-edge-connected and removes $v$ from $L$. For any $k$-cut $T$ where $v \in T$, an important observation is that $kECC(v) \subseteq T$ as well. But $kECC(u) = kECC(v)$ and so $u \in T$ too. So the invariant still holds even after removing $v$ from $L$.

In the second case, the algorithm removes $U$ from $G$. Let us denote $G' = G \setminus U$. Since the algorithm adds the endpoints of cut edges crossing $U$ to $L$, it suffices to consider a $k$-cut $T$ in $G'$ that is disjoint from the endpoints of the cut edges of $U$. By Lemma 8, $T$ was a $k$-cut in $G$. Since the changes in $L$ occur only at $U$ and neighbors of $U$, while $T$ is disjoint from both $U$ and all neighbors of $U$, we have $T \cap L \neq \emptyset$ by the induction hypothesis. ◀

▶ **Corollary 10.** *When $|L| = 1$, then $G$ is $k$-edge-connected.*

**Proof.** Otherwise, there is a partition $(A, B)$ of $V$ where $|E(A, B)| < k$. So both $A$ and $B$ are $k$-cuts in $G$. By Lemma 9, $A \cap L \neq \emptyset$ and $B \cap L \neq \emptyset$ which contradicts that $|L| = 1$. ◀

We are ready to conclude the correctness of Algorithm 1. At a high level, the algorithm finds the set $U$ and "cuts along" $U$ at Lines 7. Then, on one hand, recurse on $U$ at Line 8 and, on the other hand, continue on $V(G) \setminus U$. We say that the cut edges $E(U, V(G) \setminus U)$ are "deleted".

Now, since $U$ is the connected component of $G[kECC(u)]$ for some vertex $u$. We have that, for every edge $(x, y) \in E(U, V(G) \setminus U)$, the pair $x$ and $y$ are not $k$-edge-connected in $G$. In particular, $x$ and $y$ are not $k$-edge-connected in $G[V']$ for every $V' \subseteq V$.

**Algorithm 2** UPDATE(H, e).

**Input:** A $k$-edge-connected subgraph $H$ and an edge $e = (x, y) \in H$ to be deleted.
**Output:** The $k$-edge-connected subgraphs of $H$ after deletion.
1 $H \leftarrow H \setminus \{(x, y)\}$.
2 **return** MAIN$(H, \{x, y\})$.

Thus, the algorithm never deletes edges inside any maximal $k$-edge-connected subgraph $V_i$. Since the algorithm stops only when the remaining graph is $k$-edge-connected, the algorithm indeed returns the maximal $k$-edge-connected subgraphs of the whole graph.

### Running Time

Consider the time spent on each recursive call. Let $G'$ be the graph for which the recursive call is made and $m' = \mathrm{vol}(G')$. Every vertex is inserted to $L$ initially or as an endpoint of some removed edge, so the total number of vertices added to $L$ is $O(m')$. In each iteration, either we remove a vertex from $L$, or remove a subgraph from $G$. Hence we check pairwise $k$-edge-connectivity $O(m')$ times, so the running time of checking pairwise $k$-edge-connectivity is $O(t_q \cdot m')$. For the time of finding connected components of $k$-edge-connected components, since we spend $O(t_q \cdot \mathrm{vol}(U))$ time to find some $U$ and remove $U$ from $G$, the total cost is $O(t_q \cdot m')$. Plus, initializing the dynamic pairwise $k$-edge connectivity algorithm on $G'$ takes $O(t_p \cdot m')$ time. Thus the total running time of each recursive call is $O((t_p + t_q) \cdot m')$.

For the recursion depth, since each $U$ found has the smaller volume of the two, $\mathrm{vol}(U) \leq m'/2$. Hence the recursion depth is $O(\log m_0)$, where $n_0$ and $m_0$ are the numbers of vertices and edges of the initial graph. Thus the total running time of Algorithm 1 is $O((t_p + t_q) \cdot m_0 \log n_0)$.

By applying Theorem 5 to the initial graph $G$ and invoking Algorithm 1 on the resulting graph, the number of edges in the resulting graph is $O(kn_0 \log n_0)$, so the running time is improved to $O(m_0 + (t_p + t_q) \cdot kn_0 \log^2 n_0)$. This completes the proof of Lemma 6.

## 4 The Decremental Algorithm

Our static algorithm can be naturally extended to a decremental dynamic algorithm. To prove Theorem 2, we prove the following reduction. By combining Lemma 11 and Theorem 3, we are done.

▶ **Lemma 11.** *Suppose there is a deterministic decremental algorithm supporting pairwise $k$-edge-connectivity that has $t_p \cdot m$ total preprocessing and update time on an initial graph with $n$ vertices and $m$ edges and query time $t_q$.*

*Then there is a deterministic decremental dynamic algorithm for maintaining the maximal $k$-edge-connected subgraphs on an undirected graph of $n$ vertices and $m$ edges with $O((t_p + t_q) \cdot m \log n)$ total preprocessing and update time, and $O(1)$ query time.*

The algorithm for Lemma 11 as is follows. First, we preprocess the initial graph $G_0$ using Algorithm 1 and obtain the maximal $k$-edge-connected subgraphs $\{V_1, \ldots, V_z\}$ of $G_0$.

Next, given an edge $e$ to be deleted, if $e$ is in a maximal $k$-edge-connected subgraph $V_i$ of $G$, then we invoke UPDATE$(G[V_i], e)$ and update the set of the maximal $k$-edge-connected subgraphs of $G$; otherwise we ignore $e$. The subroutine UPDATE$(H, e)$ is described in Algorithm 2.

### Correctness

Let $H = (V', E')$ be the maximal $k$-edge-connected subgraph containing edge $(x, y)$ before deletion. It suffices to prove that Lemma 9 holds when we invoke Algorithm 1. Suppose there is a $k$-cut $C$ in $H \setminus \{(x, y)\}$ such that $C \cap \{x, y\} = \emptyset$, then $C$ is also a $k$-cut in $H$, a contradiction. Hence the correctness follows from the correctness of Algorithm 1.

### Running Time

In the case that $H \setminus \{(x, y)\}$ is still $k$-edge-connected, the running time is $t_q$. We charge this time $t_q$ to the deleted edge $(x, y)$.

Otherwise, consider the time spend on each recursive call of Main. Assume that the total volume of the subgraphs removed and passed to another recursive call in a recursive call is $\nu$. The total number of vertices added to $L$ is $O(\nu)$. In each iteration, we either remove a vertex from $L$ or remove a subgraph. Hence we check pairwise $k$-edge-connectivity $O(\nu)$ times, so the running time of checking pairwise $k$-edge-connectivity is $O(t_q \cdot \nu)$. Since we spend $O(t_q \cdot \mathrm{vol}(U))$ time to find $U$, the total cost is $O(t_q \cdot \nu)$. Plus, it takes $O((t_p + t_q) \cdot m')$ time to initialize the dynamic pairwise $k$-edge connectivity algorithm and check pairwise $k$-edge-connectivity on a graph $H'$ with $m'$ edges for the first time we invoke Main on $H'$. Also, removing all edges from $H'$ takes $t_p \cdot m'$ time. We charge $O(t_p + t_q)$ to each of the removed edges in each recursive call.

The recursion depth is $O(\log m_0)$ by Lemma 6, where $n_0$ and $m_0$ are the numbers of vertices and edges of the initial graph. Hence each edge will be charged $O(\log m_0)$ times, so the total preprocessing and update time is $O((t_p + t_q) \cdot m_0 \log n_0)$.

## References


1. Amir Abboud, Robert Krauthgamer, Jason Li, Debmalya Panigrahi, Thatchaphol Saranurak, and Ohad Trabelsi. Breaking the cubic barrier for all-pairs max-flow: Gomory-hu tree in nearly quadratic time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 – November 3, 2022*, pages 884–895. IEEE, 2022. doi:10.1109/FOCS54457.2022.00088.
2. Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, and Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 – November 1, 2013*, pages 909–918. ACM, 2013. doi:10.1145/2505515.2505751.
3. Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Veronika Loitzenbauer, and Nikos Parotsidis. Faster algorithms for computing maximal 2-connected subgraphs in sparse directed graphs. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1900–1918. SIAM, 2017. doi:10.1137/1.9781611974782.124.
4. Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2046–2065. SIAM, 2020. doi:10.1137/1.9781611975994.126.
5. Loukas Georgiadis, Giuseppe F. Italiano, Evangelos Kosinas, and Debasish Pattanayak. On maximal 3-edge-connected subgraphs of undirected graphs. *CoRR*, abs/2211.06521, 2022. doi:10.48550/arXiv.2211.06521.

6 Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. Computing vertex connectivity: New bounds from old techniques. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 462–471. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548505.

7 Wenyu Jin and Xiaorui Sun. Fully dynamic s-t edge connectivity in subpolynomial time (extended abstract). In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 861–872. IEEE, 2021. doi:10.1109/FOCS52979.2021.00088.

8 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. doi:10.1145/331605.331608.

9 Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2008. doi:10.1017/CBO9780511721649.

10 Chaitanya Nalam and Thatchaphol Saranurak. Maximal $k$-edge-connected subgraphs in weighted graphs via local random contraction. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 183–211. SIAM, 2023. doi:10.1137/1.9781611977554.ch8.

11 Thatchaphol Saranurak and Wuwei Yuan. Maximal $k$-edge-connected subgraphs in almost-linear time for small $k$, 2023. doi:10.48550/arXiv.2307.00147.

12 Heli Sun, Jianbin Huang, Yang Bai, Zhongmeng Zhao, Xiaolin Jia, Fang He, and Yang Li. Efficient k-edge connected component detection through an early merging and splitting strategy. *Knowl. Based Syst.*, 111:63–72, 2016. doi:10.1016/j.knosys.2016.08.006.

13 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

14 Mikkel Thorup. Fully-dynamic min-cut. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 224–230. ACM, 2001. doi:10.1145/380752.380804.

15 Xifeng Yan, Xianghong Jasmine Zhou, and Jiawei Han. Mining closed relational graphs with connectivity constraints. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 324–333. ACM, 2005. doi:10.1145/1081870.1081908.

16 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. I/O efficient ECC graph decomposition via graph reduction. *Proc. VLDB Endow.*, 9(7):516–527, 2016. doi:10.14778/2904483.2904484.

17 Rui Zhou, Chengfei Liu, Jeffrey Xu Yu, Weifa Liang, Baichen Chen, and Jianxin Li. Finding maximal k-edge-connected subgraphs from a large graph. In Elke A. Rundensteiner, Volker Markl, Ioana Manolescu, Sihem Amer-Yahia, Felix Naumann, and Ismail Ari, editors, *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 480–491. ACM, 2012. doi:10.1145/2247596.2247652.


## A Relationship with $k$-Edge-Connected Components

The reason why a subroutine for computing $k$-edge-connected components is not useful for computing maximal $k$-edge-connected subgraphs is as follows. Given a graph $G = (V, E)$, we can artificially create a supergraph $G' = (V' \supseteq V, E' \supseteq E)$ where the whole set $V$ is $k$-edge-connected, but the maximal $k$-edge-connected subgraphs of $G'$ will reveal the maximal $k$-edge-connected subgraphs of $G$. So given a subroutine for computing the $k$-edge-connected components of $G'$, we know nothing about the maximal $k$-edge-connected subgraphs of $G$. The construction of $G'$ is as follows.

First, we set $G' \leftarrow G$. Assume $V = \{1, 2, \ldots, n\}$. For every $1 \leq i < n$, add $k$ parallel dummy length-2 paths $(i, d_{i,1}, i+1), \ldots, (i, d_{i,k}, i+1)$. Thus $i$ and $i+1$ are $k$-edge-connected, so $V$ is $k$-edge-connected at the end. When we compute the maximal $k$-edge-connected subgraphs of $G'$, we know that we will first remove all dummy vertices $d_{i,j}$ because they all have degree 2 (assuming that $k > 2$). We will obtain $G$ and so we will obtain the maximal $k$-edge-connected subgraphs of $G$ from this process.

# Approximating Connected Maximum Cuts via Local Search

**Baruch Schieber**
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA

**Soroush Vahidi**
Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA

## Abstract

The Connected Max Cut (CMC) problem takes in an undirected graph $G(V, E)$ and finds a subset $S \subseteq V$ such that the induced subgraph $G[S]$ is connected and the number of edges connecting vertices in $S$ to vertices in $V \setminus S$ is maximized. This problem is closely related to the Max Leaf Degree (MLD) problem. The input to the MLD problem is an undirected graph $G(V, E)$ and the goal is to find a subtree of $G$ that maximizes the degree (in $G$) of its leaves. [Gandhi et al. 2018] observed that an $\alpha$-approximation for the MLD problem induces an $\mathcal{O}(\alpha)$-approximation for the CMC problem.

We present an $\mathcal{O}(\log\log|V|)$-approximation algorithm for the MLD problem via local search. This implies an $\mathcal{O}(\log\log|V|)$-approximation algorithm for the CMC problem. Thus, improving (exponentially) the best known $\mathcal{O}(\log|V|)$ approximation of the Connected Max Cut problem [Hajiaghayi et al. 2015].

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Theory of computation → Routing and network design problems; Theory of computation → Approximation algorithms analysis

**Keywords and phrases** approximation algorithms, graph theory, max-cut, local search

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.93

## 1 Introduction

The Connected Max Cut (CMC) problem takes in an undirected graph $G(V, E)$ and finds a subset $S \subseteq V$ such that the induced subgraph $G[S]$ is connected and the number of edges connecting vertices in $S$ to vertices in $V \setminus S$ is maximized. This problem is known to be NP-Hard [6]. We give an $\mathcal{O}(\log\log n)$-approximation algorithm for this problem, where $n = |V|$. This improves on the previously known $\mathcal{O}(\log n)$-approximation algorithm given in [6]. (We say that an approximation algorithm for a maximization problem achieves an $\alpha \geq 1$ approximation ratio if the value of its output is at least $\frac{1}{\alpha}$ of the maximum.)

A function $f\colon 2^V \to \mathbb{R}_+$ is *submodular* if for every $A, B \in V$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. It is not difficult to verify that the cut function is submodular and non-monotone, and thus the CMC problem is a special case of non-monotone submodular function maximization subject to connectivity constraints. To the best of our knowledge there are no nontrivial approximation algorithms for the well motivated general non-monotone submodular function maximization subject to connectivity constraints.

The CMC problem is closely related to the Max Leaf Tree (MLT) and Max Leaf Spanning Tree (MLST) problems. The undirected version of the MLT problem takes in a vertex-weighted undirected graph $G(V, E, w)$ and finds a subtree with maximum weight on the leaves. The weighted version of the MLST problem requires the output tree of the MLT instance to be a spanning tree. We note that the MLT problem can also be posed as a non-monotone submodular function maximization problem subject to connectivity constraints.


In this case the submodular function $f: 2^V \to \mathbb{R}_+$ defined on subsets of vertices of $G$ is $f(U) = \sum_{x \in \mathcal{N}(U)} w(x)$, where $\mathcal{N}(U)$ is the set of neighbors of the vertices in $U$ that are not in $U$ (and $f(\emptyset) = 0$).

Gandhi et al. [4] observed that an $\alpha$-approximation algorithm for the MLT problem implies a $4\alpha$-approximation algorithm for the (edge) weighted CMC problem. Based on this observation we consider a special case of the MLT problem we call the MLD problem. In this problem we are given an undirected graph $G(V, E)$ and the goal is to find a subtree that maximizes the degree (in $G$) of its leaves, that is, the weight of vertices in this special case of the MLT problem is their degree in $G$. Applying the observation in Gandhi et al. [4] we get that an $\alpha$-approximation algorithm for the MLD problem gives a $4\alpha$-approximation algorithm for the *unweighted* CMC problem. We present an $\mathcal{O}(\log \log n)$-approximation algorithm for the MLD problem and thus an approximation algorithm with the same quality for the CMC problem.

Our algorithm is quite simple and involves successive local improvement steps. The proof that our algorithm indeed achieves an $\mathcal{O}(\log \log n)$ approximation ratio is somewhat involved and relies on the fact that the weights of the vertices in the MLD problem instance are the degrees of these vertices in an undirected graph with no parallel edges. To prove the $\mathcal{O}(\log \log n)$ approximation ratio we prove a stronger property of our solution: we prove that the total number of edges in the graph is $\mathcal{O}(\log \log n)$ times the sum of the degrees of the leaves of the computed subtree. This bound is tight as we can construct an instance that achieves this upper bound.

## 1.1 Motivation and Prior Art

We motivate both our CMC problem and the general non-monotone submodular function maximization subject to connectivity constraints. The Max Cut problem is a fundamental combinatorial optimization problem and is one of the problems listed in Karp's seminal paper on NP-Complete problems [9]. Extending the Max Cut problem to the CMC problem is natural. We note that a similar extension of the Min Cut problem is trivial since we can always find a minimum cut that separates a connected component.

The Max Cut problem has numerous applications, e.g., in circuit layout design, statistical physics [1], and image segmentation [3, 13]. Some of these applications can be extended to the connected max cut problem. For example, Hajiaghayi et al. [6] noted that the image segmentation application with the additional requirement that the pixels are connected can be modeled as a CMC problem.

There are quite a few applications of non-monotone submodular function maximization among them: maximum facility location and optimal sensor placement, segmentation problems [10], least core value of general supermodular cost games [15], optimal marketing over social networks [7], and revenue maximization for banner advertisement [2]. Some of these applications consider functions that are defined on a vertex set of a graph and thus it is natural to consider the maximization subject to graph constraints. For example, in sensor placement we may constrain the activated sensors to be connected to enable efficient communication, or sets targeted for marketing in a social network may be required to be in the same social community.

The CMC problem was introduced by Hajiaghayi et al. in [5] (see also [6]) who gave an $\mathcal{O}(\log n)$-approximation algorithm for this problem and obtained a polynomial time approximation scheme for the CMC problem on planar graphs and on bounded genus graphs.

Lee et al. [11] generalized the problem considered in [6] in two aspects: (i) they considered more general graph constraints and not just connectivity constraints, and (ii) they allowed the graph constraints to be on a separate graph from the one in which the max cut needs

to be computed (as long as both graphs are defined on the same set of vertices). The graph constraints considered in [11] include independent set, vertex cover, dominating set and connectivity. Lee et al. [11] showed a 2 approximation for this problem on graphs with bounded treewidth. Some of these results (for constraints such as independent set, dominating set, and connectivity) can be achieved also for planar graphs, bounded genus graphs, and graphs with a fixed excluded minor. The algorithm in [11] is not combinatorial but uses an LP relaxation based on the Sherali-Adams hierarchy.

As mentioned above, Gandhi et al. [4] showed that the weighted CMC problem and the MLT problem are closely related. They obtained a constant approximation algorithm for MLT problem in the special case of $\{0,1\}$ weights, and then used a "bucketing" technique to extend it to an $\mathcal{O}(\log n)$-approximation algorithm for the general case. This implies an $\mathcal{O}(\log n)$-approximation algorithm for the weighted CMC problem which improves the $\mathcal{O}\big(\log^2 n\big)$-approximation algorithm for this problem given in [6].

The unweighted MLT problem is simply the problem of finding a subtree with the maximum number of leaves. Note that there is always a spanning tree that maximizes the number of leaves, and thus the unweighted MLT problem and the unweighted MLST problem are equivalent and both can be approximated within a factor of 2 [16]. The weighted MLST problem admits no $n^{1-\varepsilon}$ approximation, for any constant $\varepsilon > 0$ as it is equivalent to the Maximum Independent Set problem, as observed in [8]. This is in contrast to the weighted MLT problem that admits an $\mathcal{O}(\log n)$ approximation [4], and to the MLD problem, which is a special case of the weighted MLT problem in which the weight of every vertex is equal to its degree, that admits an $\mathcal{O}(\log\log n)$ approximation, as shown in this paper.

The rest of the paper is organized as follows. Section 2 gives a formal definition of our problems. Section 3 describes the local search algorithm, and Section 4 proves its approximation ratio. Section 5 has some concluding remarks and open problems. Due to space constraints some of the proofs are omitted from this abridged version of the paper.

## 2 Preliminaries and Problem Definitions

In all our problems we consider undirected graphs with no parallel edges. Let $G(V,E)$ be such a graph, with $n = |V|$, and $m = |E|$. For a subset $S \subseteq V$, define $\mathcal{N}_G(S)$ to be the subset of the edges in $E$ with exactly one endpoint in $S$. For a vertex $v \in V$, let $d_G(v)$ denote the degree of $v$ in $G$; that is, $d_G(v) = |\mathcal{N}_G(\{v\})|$. For a subset $S \subseteq V$, define $G[S]$ to be the subgraph induced by $S$. Any subset $S \subseteq V$ such that $G[S]$ is connected, defines a connected cut of size $|\mathcal{N}_G(S)|$.

A tree $T(U,F)$ is a *subtree* of graph $G$ if $T$ is a tree, $U \subseteq V$, and $F \subseteq E$. We denote the "subtree" relation by the symbol $\sqsubseteq$, namely, $T \sqsubseteq G$ indicates that $T$ is a subtree of $G$. Let $V(T)$ be the set of vertices of $T$, $L(T)$ be the set of leaves of $T$, and $I(T)$ be the set of the internal vertices of $T$; that is, $L(T) = \{v \mid v \in U \wedge d_T(v) = 1\}$ and $I(T) = V(T) \setminus L(T)$. Obviously, $T(U,F)$ is a spanning tree of $G$ iff $U = V$.

For any given vertex $r \in U$, if $T$ is *rooted* at $r$, then all the edges in $F$ are oriented from $r$ outwards; namely, for $(x,y) \in F$, vertex $x$ precedes $y$ in the (unique) path from $r$ to $y$. In this case we say that $x$ is the *parent* of $y$ and $y$ is a *child* of $x$. Our convention is to specify the tree edge connecting $x$ to its child $y$ as $(x,y)$. If vertex $x$ is on the (unique) path from $r$ to $y$ then $x$ is an *ancestor* of $y$ and $y$ is a *descendant* of $x$. We denote this relation by $x \rightsquigarrow y$. A vertex $x$ is considered an ancestor and a descendant of itself (unless specified explicitly otherwise). For a subset $U' \subseteq U$, define $\mathsf{Desc}_T(U')$ to be the set of descendants of the vertices in $U'$, and $\mathsf{LDesc}_T(U')$ to be the set of *leaf* descendants of the vertices in

$U'$. Namely, $\mathsf{Desc}_T(U') = \{y \mid \exists x \in U' \colon\ x \rightsquigarrow y\}$, and $\mathsf{LDesc}_T(U') = \mathsf{Desc}_T(U') \cap L(T)$. To simplify notation for singleton sets, we define $\mathsf{Desc}_T(u) = \mathsf{Desc}_T(\{u\})$. We assume that $n > 2$ (since the problem is trivial when $n \leq 2$), and for convenience we always root $T$ at a vertex $r$ for which $d_T(r) > 1$.

For a vertex $x \in U$, let $C_T(x)$ be the set of its children in $T$, and $T_x$ be the subtree rooted at $x$; that is, the tree induced by the set of the descendants of $x$. For any two vertices $x, y \in U$, let $LCA(x, y)$ denote the lowest ancestor of $x$ and $y$; defined as the unique vertex $u \in U$ such that $u$ is an ancestor of both $x$ and $y$, and none of the children of $u$ is an ancestor of both $x$ and $y$.

For a graph $G(V, E)$ and a tree $T(U, F)$, where $U \subseteq V$, define the *leaf degree* of $T$ in $G$, denoted $\mathsf{LD}_G(T)$, as $\mathsf{LD}_G(T) = \sum_{v \in L(T)} d_G(v)$; in words, the leaf degree of $T$ is the sum of the degrees in $G$ of the leaves of $T$.

The input for both the Connected Max Cut (CMC) problem and the Max Leaf Degree (MLD) problem is a graph $G(V, E)$. The output of the CMC problem is a connected cut of maximum size. The output of the MLD problem is a subtree $T_{\mathsf{opt}} \sqsubseteq G$ with maximum leaf degree, i.e., $T_{\mathsf{opt}} = \arg\max_{T \sqsubseteq G} \{\mathsf{LD}_G(T)\}$.

Hajiaghayi et al. [6] proved that the CMC problem is NP-Hard even when the input is restricted to planar graphs. The MLD problem is NP-Hard as well. To see this note that when the graph $G(V, E)$ is regular the MLD problem is equivalent to the unweighted MLT problem (since all vertices have the same weight), which is equivalent to the unweighted MLST problem, as noted above. The unweighted MLST problem on regular graphs was shown to be NP-Hard in [12, 14] and thus the MLD problem is also NP-Hard even when the input is restricted to regular graphs. The MLD problem is APX-Hard as well since the unweighted MLST problem on 5-regular graphs is APX-Hard [14].

Both problems have a weighted version which is not considered in this paper. In the weighted CMC problem every edge $e \in E$ has a weight and the output of the weighted CMC problem is a connected cut of maximum weight. In the MLT problem, which is the weighted MLD problem, every vertex $v \in V$ has a weight $w(v)$ and the output of the MLT problem is a subtree $T_{\mathsf{opt}} \sqsubseteq G$ with maximum leaf weight, i.e., $T_{\mathsf{opt}} = \arg\max_{T \sqsubseteq G} \left\{\sum_{v \in L(T)} w(v)\right\}$. (Recall that the MLD problem is a special case of the weighted MLT problem in which the weight of the vertices is equal to their degree.)

Since both the CMC and the MLD problems are NP-Hard they call for approximation algorithms. Both are maximization problems. We say that an approximation algorithm for a maximization problem achieves an $\alpha \geq 1$ approximation ratio if the value of its output is at least $\frac{1}{\alpha}$ of the maximum. Gandhi et al. [4] proved a relation between the approximation of these problems.

▶ **Theorem 1** ([4]). *An $\alpha$-approximation algorithm for the MLD problem implies a $4\alpha$-approximation algorithm for the CMC problem.*

We note that [4] considered the weighted CMC and MLT problems. For an edge weighted CMC instance the weight of a vertex $v$ in the corresponding MLT instance is the sum of the weights of the edges in $\mathcal{N}_G(\{v\})$. It is easy to see that the MLD problem corresponds to the unweighted CMC problem since in the MLD problem the weight of every vertex equals its degree.

# 3 The Local Search Algorithm for MLD

Given a graph $G(V, E)$, we present a local search algorithm that finds a subtree $T^* \sqsubseteq G$ such that $\mathcal{O}(\mathsf{LD}_G(T^*) \log \log n) = \max_{T \sqsubseteq G} \{\mathsf{LD}_G(T)\}$. We start by computing any spanning tree $T(V, F)$ of $G$, and root it at an arbitrary vertex $r \in V$. The solution generated by our algorithm will be a subtree of $G$, also rooted at $r$. The algorithm proceeds iteratively by performing local search improvements. Each local search improvement considers an internal vertex $v \in I(T)$ and determines whether making $v$ a leaf, while potentially removing some of the current leaves in $T$, results in an improved tree. We call such an operation a *vertex improvement* defined below. The algorithm terminates when no local search improvements are available and the current tree $T$ at this point is returned as $T^*$.

## 3.1 Vertex improvement

To define the improvements we first introduce the notion of *dependent* children. Let $T \sqsubseteq G$ be a subtree of $G$. Consider a vertex $v \in I(T)$, and let $C_T(v) \subseteq V(T)$ be the set of children of $v$ in $T$. For a vertex $x \in C_T(v)$, consider $G[I(T) \cup \{x\} \setminus \{v\}]$ – the subgraph of $G$ induced by $x$, and the vertices in $I(T)$, except for $v$. We say that $x$ is a *dependent* child of $v$ if $x$ is disconnected from $r$ in this induced subgraph. Note that a *dependent* child of $v$ depends on $v$ for its connectivity to $r$ in the subgraph $G[I(T) \cup \{x\}]$ (implying that $v$ is a separating vertex in $G[I(T) \cup \{x\}]$). If a vertex $x \in C_T(v)$ is not a dependent child then it is an *independent* child of $v$. Let $D_T(v) \subseteq C_T(v)$ be the (possibly empty) set of dependent children of $v$ in $T$. (For the root $r$ of $T$, trivially $D_T(r) = C_T(r)$.) Figure 1 illustrates a vertex $v$ and its *independent* child $x$.

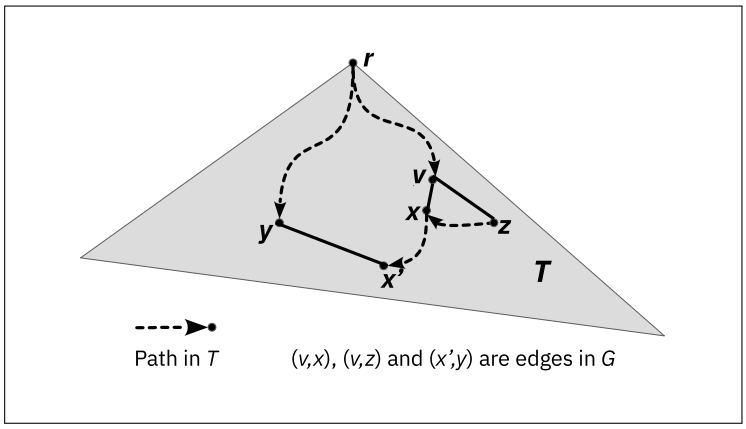



**Figure 1** Vertices $x$ and $z$ are independent children of $v$ since the paths from $z$ to $x$ and from $x$ to $r$ via $(x', y)$ use only internal vertices of $T$ and avoid $v$.

▶ **Lemma 2.** *For a subtree $T(U, F) \sqsubseteq G$ and a vertex $v \in I(T)$, if $v$ has any independent children, then there exists an independent child $x \in C_T(v) \setminus D_T(v)$ and a subtree $T'(U, F') \sqsubseteq G$, such that $C_{T'}(v) = C_T(v) \setminus \{x\}$, $L(T) \subseteq L(T') \subseteq L(T) \cup \{v, x\}$, and $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$.*

**Proof.** Consider an independent child $z \in C_T(v) \setminus D_T(v)$ (see Figure 1). By the definition of an independent child there exists a path in $G$ that connects $z$ to the root $r$ with the property that all the vertices along this path are in $I(T) \setminus \{v\}$. Follow such a path starting at $z$. It must include an edge $(x', y)$ such that $LCA(x', y)$ is an ancestor of $v$ other than $v$. (This is because $r$ is not a descendant of $v$.) Consider the first such edge along the path. For this

edge $x'$ is a descendant of $v$ (other than $v$) and $y$ is not a descendant of $v$. Adding the edge $(x', y)$ to the set $F$ of the edges of $T$ creates a cycle that consists of the tree paths from $LCA(x', y)$ to $x'$ and to $y$ and the edge $(x', y)$. Since $x'$ is a descendant of $v$ other than $v$, the path from $LCA(x', y)$ to $x'$ contains an edge that connects $v$ to one of its children. Let this child be $x \in C_T(v)$. Clearly, $x$ is also an independent child since the path from $x$ to $r$ that uses the tree path from $x$ to $x'$, the edge $(x', y)$ and the tree path from $y$ to $r$ avoids $v$.

Let $F' = F \cup \{(x', y)\} \setminus \{(v, x)\}$. (See also Figure 1.) All the vertices in $U$ are reachable from $r$ via edges of $F'$ as follows. The vertices that are not descendants of $x$ are reachable in the same way as before and the ones which are descendants of $x$ are reachable through the edge $(x', y)$ since $y$ (which is not a descendant of $x$ in $T$) is reachable from $r$ and all the descendants of $x$ are reachable from $y$ via edge $(x', y)$ and the edges in the subtree rooted at $x$ as all these edges are in $F'$. Since $|F'| = |F| = |U| - 1$ we have that $T'(U, F')$ is a tree that spans the vertex set $U$, and thus $T'(U, F') \sqsubseteq G$.

The path from $x$ to $r$ in $T'$ avoids $v$, and thus $x$ is not a descendent of $v$ in $T'$. This is the only change in the set of children of $v$, implying that $C_{T'}(v) = C_T \setminus \{x\}$. The only vertices in $U$ whose degree in $T'$ may be larger than their degree in $T$ are $y$ (always) and $x'$ (when $x' \neq x$). However, $y \in I(T)$ and when $x' \neq x$ also $x' \in I(T)$. Thus, none of the leaves of $T$ may become internal vertices in $T'$. On the other hand the degrees of $v$ and $x$ (when $x' \neq x$) in $T'$ are 1 less than their degrees in $T$ and thus they may become leaves in $T'$, implying that $L(T) \subseteq L(T') \subseteq L(T) \cup \{v, x\}$. Since $L(T) \subseteq L(T')$, we also have $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$. ◀

Given a subtree $T(U, F) \sqsubseteq G$ and a vertex $v \in I(T)$, Lemma 2 can be applied successively to obtain a tree $T'(U, F')$ in which $v$ has no independent children. This is done as follows. Start by initializing $T'$ to $T$. If $v$ has an independent child in $T'$, apply Lemma 2 to get a modified tree $T'$ that spans the same set of vertices $U$ and satisfies the conditions of the lemma. If vertex $v$ still has an independent child in the modified $T'$, then repeat the application of Lemma 2. The process is bound to terminate since the number of children of $v$ is decreased in each iteration. Note that when the process terminates $L(T) \subseteq L(T')$ and thus $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$.

The function VertexImprove$(G, T, v)$ described in Algorithm 1 gets a graph $G(V, E)$, a subtree $T(U, F) \sqsubseteq G$, and a vertex $v \in I(T)$ as parameters. It determines whether $\mathsf{LD}_G(T)$ can be improved by making $v$ a leaf. It returns a subtree $T'(U', F') \sqsubseteq G$ such that $\mathsf{LD}_G(T') \geq \mathsf{LD}_G(T)$. (In case of improvement $\mathsf{LD}_G(T') > \mathsf{LD}_G(T)$.)

**Algorithm 1** Check whether the leaf degree can be improved by making $v$ a leaf.

```
1:  function VertexImprove(G, T, v)
2:      Parameters: graph G(V, E), subtree T(U, F) ⊑ G, vertex v
3:      Returned value: subtree T' ⊑ G, s.t. LD_G(T') ≥ LD_G(T)
4:      T' ← T
5:      while ∃x ∈ C_{T'}(v) ∧ ∃(x', y) ∈ E such that x ⇝ x' and v ↛ y do
6:          Update tree T' by swapping the tree edge (v, x) with the nontree edge (x', y)
7:      if v is a leaf in T' then
8:          return T'
9:      else if d_G(v) ≤ 2LD_G(T'_v) then                ▷ no substantial improvement
10:         return T'
11:     else                          ▷ remove the subtrees rooted at the children of v
12:         return T' ← T'[(U \ Desc_{T'}(v)) ∪ {v}]
```

The function VertexImprove starts by modifying the tree $T$ to a tree $T'(U, F')$ in which $v$ has no independent children as described above. (Note that any independent child of $v$ before the modification becomes an independent child of another vertex in $I(T)$.) At this point $v$ may either be a leaf in $T'$ or an internal vertex with only dependent children. If $v$ is a leaf, then $L(T) \cup \{v\} \subseteq L(T')$ and $\mathsf{LD}_G(T') > \mathsf{LD}_G(T)$, resulting in an improvement of the leaf degree. In this case tree $T'(U, F')$ is returned by the function. If $v$ has only dependent children, then function VertexImprove checks whether removing the subtrees rooted at the children of $v$, and thus making $v$ a leaf improves the leaf degree (substantially). (See Figure 2.) Specifically, let $T'[\tilde{U}]$ be the subgraph of $T'$ induced by the vertex set $\tilde{U} = (U \setminus \mathsf{Desc}_{T'}(v)) \cup \{v\}$. Clearly, $T'[\tilde{U}]$ is connected and thus $T'[\tilde{U}] \sqsubseteq G$. Compare $\mathsf{LD}_G(T')$ and $\mathsf{LD}_G(T'[\tilde{U}])$: The only leaf in $T'[\tilde{U}]$ that is not a leaf in $T'$ is $v$. On the other hand the only leaves in $T'$ that are not in $T'[\tilde{U}]$ are the descendant leaves of $v$ in $T'$, namely, $\mathsf{LDesc}_{T'}(v)$. Let $T'_v$ be the subtree of $T'$ rooted at $v$. We get that $\mathsf{LD}_G(T'_v)$ is the sum of the degrees in $G$ of the leaves in $\mathsf{LDesc}_{T'}(v)$. Consequently, $\mathsf{LD}_G(T'[\tilde{U}]) - \mathsf{LD}_G(T') = d_G(v) - \mathsf{LD}_G(T'_v)$. It follows that if $d_G(v) > \mathsf{LD}_G(T'_v)$, then $\mathsf{LD}_G(T'[\tilde{U}]) > \mathsf{LD}_G(T')$. To achieve our approximation ratio we perform an improvement step in this case only if the resulting improvement is "substantial"; that is, only if $d_G(v) > 2\mathsf{LD}_G(T'_v)$ or equivalently $\mathsf{LD}_G(T'[\tilde{U}]) - \mathsf{LD}_G(T') > \mathsf{LD}_G(T'_v)$. In this case $T'[\tilde{U}]$ is returned by the function VertexImprove$(G, T, v)$.

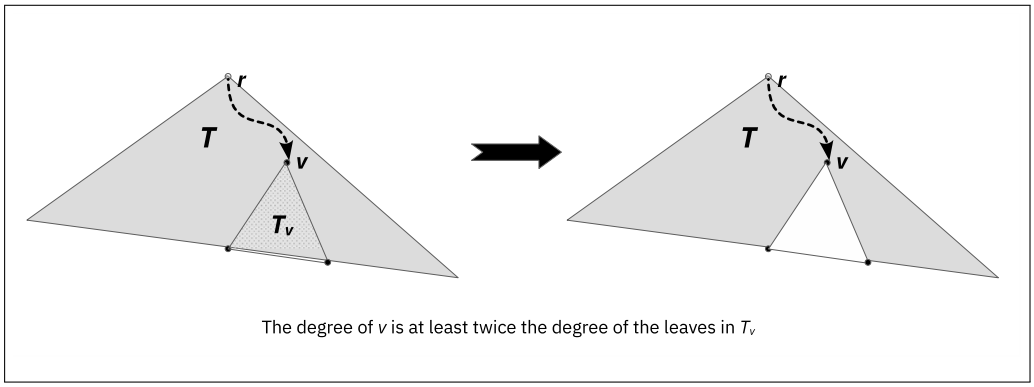



**Figure 2** Removing the children of $v$ and all their descendants.

It follows from the proof of Lemma 2 that as long as $v$ has any independent children, the condition of the While loop in Line 5 of Algorithm 1 must be satisfied. After the execution of this While loop all the vertices in $C_{T'}(v)$ must be dependent children of $v$. Note that Algorithm 1 can be implemented in polynomial time.

## 3.2 Sequencing the improvement steps

Algorithm 2 starts from an arbitrary spanning tree and calls the function VertexImprove successively, as long as tree $T$ has an internal vertex that was not considered as a candidate for improvement by this function. An internal vertex $v$ is considered as a candidate for improvement only after all its nonleaf children were considered as candidates for improvement. Such an internal vertex is guaranteed to exist whenever there are internal vertices that were not considered for improvement. The output of Algorithm 2 is the final tree, denoted $T^*$.

We prove the following properties of the tree $T^*$ that is the output of Algorithm 2.

▶ **Lemma 3.** *For every vertex $v \in I(T^*)$, $d_G(v) \leq 2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u) \leq 2\mathsf{LD}_G(T^*_v)$.*

**Algorithm 2** Local improvements.

**Input:** graph $G(V, E)$, and an arbitrary vertex $r \in V$
**Output:** a subtree $T^* \sqsubseteq G$ rooted at $r$
1: $T(V, F) \leftarrow$ a spanning tree rooted at $r$
2: $tested(v) \leftarrow$ false, $\forall v \in I(T)$
3: $tested(v) \leftarrow$ true, $\forall v \in L(T)$
4: **while** $\exists v \in I(T)$ s.t. $tested(v) =$ false **do**
5: find a vertex $v \in I(T)$ s.t. $tested(v) =$ false $\wedge$ $\forall u \in C_T(v)$ $tested(u) =$ true
6: $T \leftarrow$ VertexImprove$(G, T, v)$
7: $tested(v) \leftarrow$ true
8: **return** $T^* \leftarrow T$

**Proof.** Algorithm 2 starts with a spanning tree $T$ rooted at $r$. The tree $T$ is modified in every call to VertexImprove. Each such change may involve the removal of vertices from $T$ and conversion of some vertices from internal vertices to leaves. However, a leaf is never changed to an internal vertex, and a vertex that was removed from $T$ will never be brought back. Consider an internal vertex $v \in I(T^*)$ and the function call VertexImprove$(G, T, v)$. Fix $T$ to be the tree after this call. Vertex $v$ has to be an internal vertex of $T$ (as otherwise it would not be in $I(T^*)$). It follows that after this call $d_G(v) \leq 2\mathsf{LD}_G(T_v)$, and all the vertices in $C_T(v)$ are dependent children of $v$. Since all these vertices were children of $v$ also before the function call VertexImprove$(G, T, v)$, then by our algorithm they were considered for improvement before this call. It follows that removing any of these vertices in subsequent calls would imply also the removal of $v$, and since removed vertices are not brought back this did not happen. Thus, $D_T(v) = C_T(v) \subseteq C_{T^*}(v)$. Since $I(T^*) \subseteq I(T)$ all these vertices are also dependent children of $v$ in $T^*$ and thus $d_G(v) \leq 2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u)$. Clearly, $2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u) \leq 2\mathsf{LD}_G(T^*_v)$. ◀

▶ **Lemma 4.** *There are no edges connecting vertices in $I(T^*)$ to vertices outside $T^*$.*

**Proof.** We show that the following assertion holds throughout the computation of Algorithm 2: there are no edges connecting vertices in $I(T)$ to vertices outside $T$. The lemma follows since $T^*$ is the final value of $T$. The assertion holds at the start of Algorithm 2 since initially $T$ is a spanning tree and thus there are no vertices outside $T$. Suppose that the assertion holds for $T$ until the function call VertexImprove$(G, T, v)$. We need to show that it holds also after this call. This clearly holds in case the vertices of $T$ are not changed in this call. The only nontrivial case is when some vertices are removed from $T$ in this call; that is, Line 12 in Algorithm 1 is executed. Recall that the removed vertices are all the descendants of $v$ (other than $v$ itself) in the tree $T'$, and that in tree $T'$ all the children of $v$ are dependent children; i.e., $D_{T'}(v) = C_{T'}(v)$. This implies that none of the vertices in $desc_{T'}(v)$ are connected to an internal vertex of $T'[(U \setminus \mathsf{Desc}_{T'}(v)) \cup \{v\}]$ (which is the returned value of $T$) as this will imply that at least one of the vertices in $C_{T'}(v)$ is independent. Thus, the assertion holds for $T$ also after the call. ◀

It is not difficult to see that Algorithm 2 can be implemented in polynomial time. We remark that the efficiency of the algorithm can be improved by avoiding the search for a vertex $v$ for which $tested(v) =$ false $\wedge$ $\forall u \in C_T(v)$ $tested(u) =$ true (Line 5). This can be done by computing a postorder of the vertices of the initial tree $T$ (which is a spanning tree) and following this order when the vertices are considered for improvement. Recall that a vertex $v$ appears in a postorder after all its descendants. So, this is the case for the initial

tree. Now, suppose that Algorithm 2 follows the vertices in this order, and a vertex $v$ is considered for improvement. At this point all the descendants of $v$ in the initial tree were considered already. Any descendant of $v$ at this point that was not a descendant of $v$ in the original tree became a descendant as a result of a move of vertices in a function call to VertexImprove. The only vertices that are moved in this function call must have been considered for improvement already since they are the descendant of the vertex considered for improvement in the call to VertexImprove. Thus, when a vertex $v$ is considered for improvement all its descendants were already considered for improvement.

## 4 The Approximation Ratio of the Algorithm

Recall that $T^*$ is the tree computed by Algorithm 2. We start by proving that the number of edges with at least one endpoint in $V(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. We prove this bound separately for tree edges and nontree edges. Next, we show that the total number of edges of $G$ outside $T^*$ is also $\mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. It follows that $m = \mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. Since for any subtree $S$ of $G$, $\mathsf{LD}_G(S) \leq 2m$, we get that for any subtree $S$ of $G$, $\mathsf{LD}_G(S) = \mathcal{O}(\mathsf{LD}_G(T^*)\log\log n)$. This implies that Algorithm 2 is an $\mathcal{O}(\log\log n)$-approximation algorithm for both the MLD and the respective CMC problems.

### 4.1 Bounding the number of tree edges in $T^*$

We show that the number of tree edges is $\mathcal{O}(\mathsf{LD}_G(T^*))$. For this we upper bound the number of vertices in $T^*$ which also bounds the number of tree edges in $T^*$. Clearly, the number of leaves is $|L(T^*)|$. Also, the number of vertices of degree at least 3 in $T^*$ is bounded by $|L(T^*)| - 1$. We still need to bound the number of vertices of degree 2 in $T^*$. To prove this bound we apply a lemma from [6] that states that WLOG it can be assumed that the input graph to the CMC problem and thus also to the respective MLD problem does not contain a path of three vertices of degree 2.

▶ **Lemma 5** ([6]). *WLOG it can be assumed that the input graph to the CMC problem and thus also to the respective MLD problem does not contain a path of three vertices of degree 2.*

We note that [6] considered the slightly more general $\{0,1\}$-weighted case rather than the unit weight case considered here.

For the remainder of our analysis we color the edges of $T^*$ in two colors *red* and *blue*. For $v \in I(T^*)$, a tree edge $(v,u)$ is colored red if $u$ is a dependent child of $v$ and colored blue otherwise. Trivially, all the edges outgoing from $r$ are red. By Lemma 3 every vertex $v \in I(T^*)$ has at least one outgoing red edge. We now bound the number of degree 2 vertices in $T^*$.

▶ **Lemma 6.** *The number of degree 2 vertices in $T^*$ is bounded by $8|L(T^*)| + 3\mathsf{LD}_G(T^*)$.*

**Proof.** Consider the subgraph of $T^*$ induced by its degree 2 vertices. This subgraph is a collection of paths. The endpoint of each such path is the parent of a distinct vertex of degree other than 2 in $T^*$. Hence, the number of these paths is bounded by the number of vertices with degree other than 2 in $T^*$. As stated above, the number of vertices of degree 1 in $T^*$ is $|L(T^*)|$, and the number of vertices of degree 3 or more is bounded by $|L(T^*)| - 1$. Thus, the total number of these paths is bounded by $2|L(T^*)| - 1$. Consider the paths of length (in edges) at most 3. Let $n_3$ be the number of these paths. The number of vertices of degree 2 on these paths is bounded by $4n_3$.

Consider the rest of the paths, and denote their number by $n_{\geq 4}$. Let $v_1, v_2, v_3, \ldots, v_\ell$, for $\ell \geq 5$, be such a path. Note that the edges $(v_i, v_{i+1})$, for $i \in [1..\ell-1]$, must be red since every internal vertex must have at least one dependent child and $v_{i+1}$ is the only child of $v_i$; namely, $C_{T^*}(v_i) = D_{T^*}(v_i) = \{v_{i+1}\}$. By Lemma 5 $G$ does not contain a path of three vertices of degree 2. Thus, at least one out of every three consecutive vertices on the path $v_2, \ldots, v_{\ell-1}$ must have degree at least 3 in $G$. Hence, at least $\lfloor \frac{\ell-2}{3} \rfloor$ vertices on the path $v_2, \ldots, v_{\ell-1}$ must have degree at least 3 in $G$. Let $v_i$ be one of these vertices, and let $y \notin \{v_{i-1}, v_{i+1}\}$ be a vertex connected to $v_i$ in $G$ ($y \notin \{v_{i-1}, v_{i+1}\}$ because $G$ has no parallel edges). By Lemma 4 vertex $v_i$ is not connected to vertices outside $T^*$, and hence $y \in V(T^*)$. We claim that $y$ must be a leaf. To obtain a contradiction assume that this is not the case and consider $LCA(v_i, y)$. If $LCA(v_i, y) \neq v_i$, then it must be an ancestor of $v_{i-1}$ but not $v_{i-1}$ itself (since $v_i$ is the only child of $v_{i-1}$). In this case $v_i$ cannot be a dependent child of $v_{i-1}$ because the path from $v_i$ to $y$ using edge $(v_i, y)$ and then up the tree to $LCA(v_i, y)$ avoids $v_{i-1}$. If $LCA(v_i, y) = v_i$, then $y$ must be a descendant of $v_{i+1}$ but not $v_{i+1}$ itself. In this case the only child of $v_{i+1}$ cannot be a dependent child because the path from this child down the tree to $y$ and then to $v_i$ using edge $(v_i, y)$ avoids $v_{i+1}$.

It follows that at least $\lfloor \frac{\ell-2}{3} \rfloor \geq \frac{\ell-4}{3}$ vertices on every path $v_1, v_1, v_2, \ldots, v_\ell$ of degree 2 vertices in $T^*$, where $\ell \geq 4$, are connected to leaves in $L(T^*)$. Thus, each one of the $n_{\geq 4}$ paths of length $\ell \geq 4$ of degree 2 vertices in $T^*$ contributes at least $\frac{\ell-4}{3}$ to the leaf degree of $T^*$, which equals $\mathsf{LD}_G(T^*)$. Summing this over all such paths implies that the total number of vertices of degree 2 on these paths is bounded by $3\mathsf{LD}_G(T^*) + 4n_{\geq 4}$. Since $n_3 + n_{\geq 4} < 2|L(T^*)|$ the number of degree 2 vertices is bounded by $8|L(T^*)| + 3\mathsf{LD}_G(T^*)$. ◀

We conclude the following theorem.

▶ **Theorem 7.** *The number of tree edges in $T^*$ is bounded by $10|L(T^*)| + 3\mathsf{LD}_G(T^*) = \mathcal{O}(\mathsf{LD}_G(T^*))$.*

## 4.2 Bounding the number of nontree edges with at least 1 endpoint in $V(T^*)$

The number of nontree edges with at least one endpoint in $L(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*))$. Thus, from now on we concentrate on bounding the number of nontree edges with at least one endpoint in $I(T^*)$, and the other not in $L(T^*)$. We have already established in Lemma 4 that a vertex in $I(T^*)$ cannot be connected to a vertex outside $T^*$. It follows that it suffices to bound the number of nontree edges with both endpoints in $I(T^*)$.

Recall that for $v \in I(T^*)$, a tree edge $(v, u)$ is colored red if $u$ is a dependent child of $v$ and colored blue otherwise. For a vertex $v \in I(T^*)$, let $B_{T^*}(v)$ be the subset of vertices of $T^*$ that can be reached from $v$ by paths that start at a red edge outgoing from $v$ and then use only blue edges. In other words, $B_{T^*}(v)$ consists of all the dependent children of $v$ and all the vertices that are reachable from these dependent children using only blue edges. (Notice that $v \notin B_{T^*}(v)$.) In the next lemma we prove that an internal vertex $x \in B_{T^*}(v)$ cannot be connected to an internal vertex of $T^*$ that is neither $v$ nor in $B_{T^*}(v) \cup B_{T^*}(x)$.

▶ **Lemma 8.** *If an internal vertex $x \in B_{T^*}(v)$ is connected by a nontree edge to a vertex $y \in I(T^*)$, then $y \in \{v\} \cup B_{T^*}(v) \cup B_{T^*}(x)$.*

**Proof.** To obtain a contradiction assume that $x$ is connected to a vertex $y \in I(T^*) \setminus (\{v\} \cup B_{T^*}(v) \cup B_{T^*}(x))$. We consider 4 cases.

**Case 1.** $y$ is a descendant of $x$. Follow the path from $x$ to $y$ in $T^*$. The vertex immediately after $x$ along this path must be in $B_{T^*}(v) \cup B_{T^*}(x)$. By our assumption $y \notin B_{T^*}(v) \cup B_{T^*}(x)$. Thus, the path from $x$ to $y$ in $T^*$ must leave either $B_{T^*}(v)$ or $B_{T^*}(x)$. Note that for any vertex $u \in I(T^*)$, any tree edge connecting a vertex in $B_{T^*}(u)$ to a vertex outside $B_{T^*}(u)$ must be red. It follows that the path from $x$ to $y$ in $T^*$ must include a red edge $(x', x'')$, where $x' \neq x$ is a descendant of $x$. Note that vertex $x''$ is a dependent child of $x'$ since $(x', x'')$ is red. This is a contradiction since the path from $x''$ down the tree to $y$, followed by the edge $(x, y)$, and the path from $x$ down the tree to the parent of $x'$ avoids $x'$, and thus $x''$ cannot be a dependent child of $x'$.

**Case 2.** vertex $y$ is a descendant of a vertex $z \in B_{T^*}(v)$, where $z \neq x$, and $y$ is not a descendant of $x$. Since $y \notin B_{T^*}(v)$ the path from $z$ to $y$ in $T^*$ must include a red edge $(z', z'')$, where $z' \in B_{T^*}(v)$. In this case $z''$ is an ancestor of $y$, and thus $z''$ is connected to $x$ through the edge $(x, y)$. To obtain a contradiction we show that there exists a path from $x$ to the parent of $z'$ that avoids the vertex $z'$. This implies that $z''$ cannot be a dependent child of $z'$ which is a contradiction. Consider $LCA(x, z')$. By our assumption in this case $x$ is not an ancestor of $y$ and thus it also cannot be an ancestor of $z'$. Thus, $LCA(x, z') \neq x$. In case $LCA(x, z') \neq z'$ the path from $x$ to the parent of $z'$ goes from $x$ to $LCA(x, z')$ and then down the tree to the parent of $z'$. In case is $LCA(x, z') = z'$ we must have that $z'$ is an ancestor of $x$. Since both $x$ and $z'$ are in $B_{T^*}(v)$, all the edges on the path in $T^*$ from $z'$ to $x$ are blue. But then the ancestor of $x$ which is the child of $z'$ is not a dependent child of $z'$, and thus there is a path from $x$ to the parent of $z'$ that goes through this child and avoids $z'$.

In the remaining two cases vertex $y$ is not a descendant of any vertex in $B_{T^*}(v)$.

**Case 3.** Vertex $y$ is not a descendant of $v$. In this case there is a path from $y$ to the parent of $v$ that avoids $v$. Let $x'$ be the child of $v$ that is an ancestor of $x$. Since $x \in B_{T^*}(v)$, the tree edge $(v, x')$ is colored red, and vertex $x'$ is a dependent child of $v$ but this is a contradiction since the path from $x'$ to $x$ and then through the edge $(x, y)$ to the parent of $v$ avoids $v$.

**Case 4.** Vertex $y$ is a descendant of $v$. Consider the first edge $(v, y')$ on the path from $v$ to $y$ in $T^*$. Since $y$ is not a descendant of any vertex in $B_{T^*}(v)$ this edge must be colored blue, and thus $y'$ is an independent child of $v$ and there exists a path from $y'$ to the parent of $v$ that avoids $v$. As in the previous case, let $x'$ be the child of $v$ that is an ancestor of $x$. The tree edge $(v, x')$ is colored red, and vertex $x'$ is a dependent child of $v$. This is a contradiction since the path from $x'$ that goes down the tree to $x$, then to $y$ through the edge $(x, y)$, then up the tree to $y'$, and then from $y'$ to the parent of $v$, avoids $v$. ◀

Consider a vertex $x \in V(T^*)$ that is not the root $r$. Since all the edges outgoing from $r$ are red, there must be at least one red edge $(v, v')$ on the path up the tree from $x$ to $r$. Thus there exists a vertex $v \in I(T^*)$ such that $x \in B_{T^*}(v)$. Note that $v$ is the unique ancestor of $x$ with the property that the (unique) path from $v$ to $x$ in $T^*$ starts in a red edge and then uses only blue edges. It follows that any two sets $B_{T^*}(v)$ and $B_{T^*}(u)$ are disjoint. Thus, the sets $B_{T^*}(v)$ form a partition of the vertices in $T^*$ (excluding $r$). This implies that the total number of edges of the form $(v, x)$, where $v \in I(T^*)$ and $x \in B_{T^*}(v) \cap I(T^*)$ is bounded by $|I(T^*)|$ (since there are no parallel edges). Clearly, $|I(T^*)|$ is bounded by the number of tree edges, and thus by Theorem 7 $|I(T^*)| = \mathcal{O}(\mathsf{LD}_G(T^*))$.

It follows from Lemma 8 that we are left with bounding the number of nontree edges $(x, y)$, where both $x$ and $y$ are in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$. To bound the number of these edges we modify the tree $T^*$ to a tree $\tilde{T}$ that spans the same set of vertices as $T^*$ in which all the vertices in $B_{T^*}(v)$ become the children of $v$. Note that unlike the tree $T^*$, the tree $\tilde{T}$ is not a subtree of $G$.

### 4.2.1 The modification of $T^*$ to $\tilde{T}$

Tree $\tilde{T}$ is obtained from $T^*(U, F)$ by making all the vertices in $B_{T^*}(v)$ the children of $v$, for every $v \in I(T^*)$. Formally, for every $v \in I(T^*)$, edge $(v, x)$ is a tree edge of $\tilde{T}$ iff $x \in B_{T^*}(v)$. To illustrate the definition, consider a vertex $x \in B_{T^*}(v)$ that is not a child of $v$ in $T^*$, and let $y$ be the parent of $x$ in $T^*$. Note that the edge $(y, x)$ is a blue edge and $x$ is thus an independent child. We remove the edge $(y, x)$ and instead add the edge $(v, x)$, making $v$ the new parent of $x$. Let $\tilde{T}$ be the resulting tree. It is not difficult to see that $\tilde{T}$ is indeed a tree that spans the set $U$. This is because the parent of any vertex $v$ in $\tilde{T}$ is an ancestor of its parent in $T^*$.

▶ **Lemma 9.** *Any two vertices $\{x, y\} \subseteq B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$ are siblings in $\tilde{T}$ and share $v$ as their parent.*

**Proof.** If both $x$ and $y$ are independent children in $T^*$ then after the modification both become children of $v$. The only other vertices in $B_{T^*}(v) \cap I(T^*)$ are the dependent children of $v$ and they remain children of $v$ also in $\tilde{T}$. ◀

Recall that we need to bound the number of nontree edges of the form $(x, y)$, where both $x$ and $y$ are in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$. This is equivalent to bounding the number of the edges of $G$ that connect *siblings* in $I(\tilde{T})$.

▶ **Lemma 10.** *For any vertex $v \in I(T^*)$, if $x$ is a descendant of a dependent child of $v$ in $T^*$ then $x$ is a descendant of $v$ in $\tilde{T}$.*

By Lemma 3 for every vertex $v \in I(T^*)$, $d_G(v) \leq 2\sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u)$. By Lemma 10 any leaf descendant of a vertex $u \in D_{T^*}(v)$ is a descendant of $v$ in $\tilde{T}$ and thus we have $d_G(v) \leq 2\mathsf{LD}_G(\tilde{T}_v)$. As shown above, to bound the number of nontree edges with both endpoints in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$, it suffices to bound the number of edges that connect siblings in $I(\tilde{T})$. For this we prove the following slightly more general theorem, where instead of considering the leaf degree, we assume general weights on the leaves of $\tilde{T}$. Associate a weight $w(\ell) \geq 1$ with every leaf $\ell \in L(\tilde{T})$. For a vertex $v \in I(\tilde{T})$, let $w(\tilde{T}_v)$ denote the total weight of the leaves of $\tilde{T}_v$, which is the subtree rooted at $v$. Also define $w(\tilde{T}) = w(\tilde{T}_r)$. The following theorem (proven below) may be interesting on its own.

▶ **Theorem 11.** *Let $\tilde{T}$ be a rooted tree with an integral weight $w(\ell) \geq 1$ associated with every leaf $\ell \in L(\tilde{T})$, and let $\tilde{G}(I(\tilde{T}), \tilde{E})$ be a graph with no parallel edges that all of its edges connect siblings in $I(\tilde{T})$. If for every $v \in I(\tilde{T})$, $d_{\tilde{G}}(v) \leq 2w(\tilde{T}_v)$, then $|\tilde{E}| = \mathcal{O}\big(w(\tilde{T}) \log\log w(\tilde{T})\big)$.*

To prove this theorem we further modify $\tilde{T}$ to $\hat{T}$ in which the degree of every internal vertex is lower and upper bounded by a constant times the weight of the leaves of its rooted subtree.

### 4.2.2 The modification of $\tilde{T}$ to $\hat{T}$

We obtain $\hat{T}$ from $\tilde{T}$ by removing some of the internal vertices of $\tilde{T}$. While doing so we maintain that the number of edges connecting siblings in $I(\hat{T})$ remains the same as the number of edges connecting siblings in $I(\tilde{T})$.

Initially, $\hat{T}$ is set to $\tilde{T}$ and $\hat{G}$ is set to $\tilde{G}$. Next, we traverse the vertices of $I(\hat{T}) \setminus \{r\}$ top down starting from the children of the root $r$. When a vertex $v$ is traversed we check whether $v$ has any children that are internal vertices, and if so whether $d_{\hat{G}}(v) < w(\hat{T}_v)$. (Recall that by the conditions of Theorem 11, initially, $d_{\hat{G}}(v) \leq 2w(\hat{T}_v)$.) If this is the case we execute the following modification steps.

**Step 1.** Remove $v$ from $\hat{T}$ and connect all the children of $v$ that are internal vertices to the parent of $v$. Denote the set of these children by $C$.

**Step 2.** Pick a vertex $x \in C$ and connect $x$ to all the leaves of $\hat{T}$ that were children of $v$ before its removal.

**Step 3.** For every edge $(v, y)$ in $\hat{G}$ that connected $v$ before its removal to a sibling in $\hat{T}$, find a vertex $z \in C$ such that $d_{\hat{G}}(z) < 3w(\hat{T}_z) - 1$, and add the edge $(z, y)$ to $\hat{G}$ in lieu of the edge $(v, y)$. Lemma 12 implies that such a vertex $z$ always exists.

▶ **Lemma 12.** *After Step 2 of the modification steps*

$$d_{\hat{G}}(v) + \sum_{z \in C} d_{\hat{G}}(z) < 3 \sum_{z \in C} w(\hat{T}_z). \tag{1}$$

Lemma 12 implies that for each of the $d_{\hat{G}}(v)$ edges that connected $v$ to its siblings we can find in Step 3 a vertex $z \in C$ such that $d_{\hat{G}}(z) < 3w(\hat{T}_z) - 1$. (Recall that the weights are integral.)

▶ **Lemma 13.** *After traversing all the vertices of $\hat{T}$ the following properties are satisfied.*

**Property 1.** *The graph $\hat{G}(I(\hat{T}), \hat{E})$ has no parallel edges and all of its edges connect siblings in $I(\hat{T})$.*

**Property 2.** *The number of edges of $\hat{G}$ is the same as the number of edges of $\tilde{G}$.*

**Property 3.** *For every internal vertex $v \in I(\hat{T}) \setminus \{r\}$ that is a parent of a nonleaf vertex in $\hat{T}$, $w(\hat{T}_v) \le d_{\hat{G}}(v)$.*

**Property 4.** *For every internal vertex $v \in I(\hat{T}) \setminus \{r\}$, $d_{\hat{G}}(v) \le 3w(\hat{T}_v)$.*

### 4.2.3 Bounding the number of edges connecting siblings in $\hat{T}$

To bound the number of edges connecting siblings in $\hat{T}$ we prove the following theorem.

▶ **Theorem 14.** *Let $\hat{T}$ be a rooted tree with an integral weight $w(\ell) \ge 1$ associated with every leaf $\ell \in L(\hat{T})$, and let $\hat{G}(I(\hat{T}), \hat{E})$ be a graph with no parallel edges that all of its edges connect siblings in $I(\hat{T})$. If for every $v \in I(\hat{T})$ that is a parent of a nonleaf vertex in $\hat{T}$, $w(\hat{T}_v) \le d_{\hat{G}}(v)$, and for every $v \in I(\hat{T})$, $d_{\hat{G}}(v) \le 3w(\hat{T}_v)$, then $|\hat{E}| = \mathcal{O}\Big(w(\hat{T}) \log\log w(\hat{T})\Big)$.*

To prove Theorem 14 we use the following simple inequality that holds for undirected graphs with no parallel edges.

▶ **Lemma 15.** *Let $H(V, E)$ be an undirected graph with no parallel edges and $m = |E|$. Let $U \subseteq V$ be the subset of vertices of degree at most $4\sqrt{m}$. Then, $\sum_{v \in U} d_H(v) > \frac{7}{8}m$.*

**Proof of Theorem 14.** To bound the number of edges of $\hat{G}$ we use a charging scheme. For a vertex $v \in I(\hat{T})$ that is a parent of at least 2 nonleaf vertices in $\hat{T}$, let $\hat{G}_v = \hat{G}[C_{\hat{T}}(v) \cap I(\hat{T})]$; that is, the subgraph of $\hat{G}$ induced by the children of $v$ that are internal vertices. Note that $\hat{G}$ is the union of all these subgraphs since $\hat{G}$ only contains edges that connect siblings in $I(\hat{T})$. Let $m_v$ be the number of edges in this subgraph. "Charge" these $m_v$ edges to the subset of children of $v$ whose degree in $\hat{G}_v$ and thus also in $\hat{G}$ is at most $4\sqrt{m_v}$. By Lemma 15 the sum of the degrees of all these children is greater than $\frac{7}{8}m_v$. Thus it suffices to charge each such child $x$ of $v$ the amount $\frac{8}{7} \cdot d_{\hat{G}}(x)$ to cover for all the $m_v$ edges.

To bound the total number of edges we need to sum the charges of all the vertices. We distinguish between charged vertices that are parents of a nonleaf vertex and those with only leaf children. There are 3 cases based on the characteristics of the charged vertex $x \in C_{\hat{T}}(v) \cap I(\hat{T})$.

**Case 1.** The charged vertex $x$ has only leaf children. By the conditions of the theorem $d_{\hat{G}}(x)$ is upper bounded by 3 times the weight of its leaf children. Since the leaf children of any two vertices are disjoint we get that the total amount charged to vertices with only leaf children is upper bounded by $\frac{8}{7} \cdot 3w(\hat{T})$.

**Case 2.** The charged vertex $x$ is a parent of a nonleaf vertex and $d_{\hat{G}}(x) \leq 48 \cdot 3 = 144$. The amount charged to all such vertices is upper bounded by $\frac{8}{7} \cdot 144$ times the number of vertices in $I(\hat{T})$ that have siblings. The number of internal vertices in $\hat{T}$ that have siblings (and thus are children of a vertex whose degree is at least 3 in $\hat{T}$) is bounded by the number of leaves of $\hat{T}$. Note that $\mathcal{O}\Big(L(\hat{T})\Big) = \mathcal{O}\Big(w(\hat{T})\Big)$ since for every leaf $\ell \in L(\hat{T})$, $w(\ell) \geq 1$. We conclude that the amount charged to all such vertices is $\mathcal{O}\Big(w(\hat{T})\Big)$.

**Case 3.** The charged vertex $x$ is a parent of a nonleaf vertex and $d_{\hat{G}}(x) > 144$. By our construction since $x$ is a parent of a nonleaf vertex we have $w(\hat{T}_x) \geq \frac{1}{3} d_{\hat{G}}(x) > 48$ and $d_{\hat{G}}(x) > w(\hat{T}_x)$. Recall that the number of edges in $\hat{G}_v$ is $m_v$. Since $d_{\hat{G}}(x) \leq 4\sqrt{m_v}$ we have also $w(\hat{T}_x) < 4\sqrt{m_v}$. On the other hand we claim that $w(\hat{T}_v) \geq \frac{2}{3} m_v$. This holds since for every child $y \in C_{\hat{T}}(v) \cap I(\hat{T})$ we have $d_{\hat{G}}(y) \leq 3w(\hat{T}_y)$ summing over all such children of $v$ we get that the sum of all degrees, which is $2m_v$, is bounded by 3 times the sum of the weights of the subtrees rooted at all these children. The sum of these weights is upper bounded by the weight of the subtree rooted at their parent $v$ and thus we have $2m_v \leq 3w(\hat{T}_v)$. Combining the two inequalities (i) $w(\hat{T}_x) < 4\sqrt{m_v}$ and (ii) $w(\hat{T}_v) \geq \frac{2}{3} m_v$, we get $w(\hat{T}_v) \geq \frac{2}{3} m_v = \frac{1}{24} \left(4\sqrt{m_v}\right)^2 > \frac{1}{24} \left(w(\hat{T}_x)\right)^2 > 2^2 \cdot 24$ (since $w(\hat{T}_x) > 48$), and thus for any ancestor $y$ of $v$ it also holds that $w(\hat{T}_y) > \frac{1}{24} \left(w(\hat{T}_x)\right)^2$. Note that $\frac{1}{24} \cdot \left(2^{2^{i-1}} \cdot 24\right)^2 = 2^{2^i} \cdot 24$. Thus, applying the inequality recursively on all the ancestors of $x$ that are also charged, implies that if $i$ ancestors of the charged vertex $x$ are also charged, then the weight of the subtree rooted at the highest such ancestor is at least $2^{2^i} \cdot 24$. This implies that the maximum number of such vertices that are charged along any path from the root to a leaf is $\mathcal{O}\Big(\log\log w(\hat{T})\Big)$.

To sum the total amount charged to vertices with a nonleaf child and degree in $\hat{G}$ greater than 144, we consider them in "layers". The first layer consists of all vertices $x$ such that (1) $x$ has a nonleaf child and $d_{\hat{G}}(x) > 144$, (2) $x$ was charged, and (3) none of the ancestors of $x$ (other than $x$) were charged. Similarly, layer $i > 1$ consists of all vertices $x$ such that (1) $x$ has a nonleaf child and $d_{\hat{G}}(x) > 144$, (2) $x$ was charged, and (3) $x$ has an ancestor (other than $x$) that belongs to layer $i - 1$. The number of layers is bounded by the maximum number of vertices that are charged along any path from the root to a leaf which is $\mathcal{O}\Big(\log\log w(\hat{T})\Big)$. By the construction of layers if vertices $x$ and $y$ are in the same layer then $\hat{T}_x$ and $\hat{T}_y$ are disjoint. Since the charge of a vertex $x$ that is a parent of a nonleaf vertex is $\frac{8}{7} \cdot d_{\hat{G}}(x) \leq \frac{8}{7} \cdot 3w(\hat{T}_x)$ the total amount charged to the vertices in the same layer is $\frac{8}{7} \cdot 3w(\hat{T})$. The proof follows since there are $\mathcal{O}\Big(\log\log w(\hat{T})\Big)$ layers. ◀

We note that the bound Theorem 14 is tight. We show a family of trees and associated graphs that achieve the upper bound in Theorem 14. Let $S_1$ be a tree of height 3 with 5 vertices: a root, 2 children of the root, and 2 leaves, each of which is a single child of a child of the root. Assign a weight of 1 to each of the 2 leaves. The graph $\hat{G}_1$ consists of a single edge connecting the 2 children of the root. For $i > 1$, the tree $S_i$ is given by taking $2^{2^{i-1}}$ copies of

$S_{i-1}$ and connecting them to a common root. The graph $\hat{G}_i$ includes $2^{2^{i-1}}$ copies of $\hat{G}_{i-1}$ associated with the copies of $S_{i-1}$. In addition, partition the $2^{2^{i-1}}$ children of the root of $S_i$ into two sets of size $2^{2^{i-1}-1}$ and add the edges of the complete bipartite graph connecting the vertices in these 2 sets. The number of these edges is $\left(2^{2^{i-1}-1}\right)^2 = 2^{2^i-2}$. Figure 3 shows the tree $S_2$ and the associated graph $\hat{G}_2$. It can be verified that the construction obeys the conditions in Theorem 14. It can be shown by a simple induction that the number of leaves in $S_i$ is $2^{2^i-1}$ and the number of edges in $\hat{G}_i$ is $i \cdot 2^{2^i-2}$ which achieves the bound tightly.

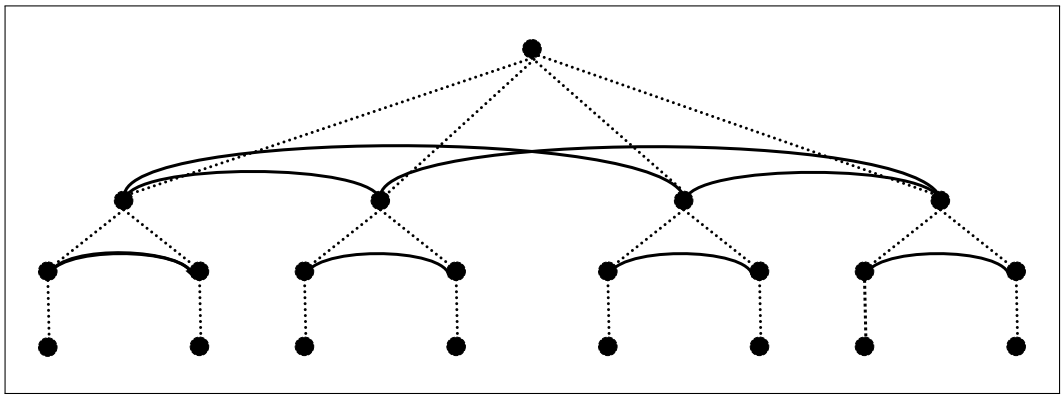


**Figure 3** The tree $S_2$ and the associated graph $\hat{G}_2$. Tree edges are dashed and graph edges are solid.

We use Theorem 14 to prove Theorem 11.

**Proof of Theorem 11.** The number of edges of $\tilde{G}$ connecting siblings in $I(\tilde{T})$ is the same as the number of edges of $\hat{G}$ connecting siblings in $I(\hat{T})$ (see Property 2 above). Also, by our modification process $w(\tilde{T}) = w(\hat{T})$. By Properties 3 and 4 above the tree $\hat{T}$ and the graph $\hat{G}$ satisfy the conditions of Theorem 14. It follows that the number of edges connecting siblings in $I(\hat{T})$ and thus also in $I(\tilde{T})$ is $\mathcal{O}\big(w(\tilde{T}) \log\log w(\tilde{T})\big)$. ◀

The analysis in Sections 4.1 and 4.2 implies the following theorem.

▶ **Theorem 16.** *The number of edges with at least one endpoint in $V(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*) \log\log n)$.*

**Proof.** Theorem 7 implies that the number of tree edges in $T^*$ is $\mathcal{O}(\mathsf{LD}_G(T^*))$. Clearly, the number of edges with one endpoint in $L(T^*)$ is $\mathcal{O}(\mathsf{LD}_G(T^*))$. As shown earlier the number of edges connecting a vertex $v$ to vertices in $B_{T^*}(v) \cap I(T^*)$ is also $\mathcal{O}(\mathsf{LD}_G(T^*))$. By Lemma 8 the only other edges with at least one endpoint in $V(T^*)$ are edges connecting 2 vertices in $B_{T^*}(v) \cap I(T^*)$, for some $v \in I(T^*)$. By Lemma 3 for every vertex $v \in I(T^*)$, $d_G(v) \leq 2 \sum_{u \in D_{T^*}(v)} \mathsf{LD}_G(T^*_u)$. This implies that after modifying $T^*$ to $\tilde{T}$ the conditions of Theorem 11 hold with $w(\ell) = d_G(\ell)$, for every leaf $\ell \in L(T^*)$. The theorem follows since $\log\log|E(G)| = \mathcal{O}(\log\log n)$. ◀

## 4.3 Bounding the number of edges outside $T^*$

The next theorem bounds the number of edges in $G[V \setminus V(T^*)]$.

▶ **Theorem 17.** *The number of edges in $G[V \setminus V(T^*)]$ is $\mathcal{O}(\mathsf{LD}_G(T^*) \log\log n)$.*

The proof of the theorem is omitted due to space constraints. It has two parts. First, we prove that there exists some constant $c$ such that the number of edges added to $G[V \setminus V(T^*)]$ when the subtrees rooted at the children of a vertex $v$ are removed from the tree is at most $c \cdot \mathsf{LD}_G(T'_v) \log\log n$. Second, we show that the total number of edges in $G[V \setminus V(T^*)]$ is bounded by $c \cdot \mathsf{LD}_G(T^*) \log\log n$ using the fact that that the subtrees rooted at the children of a vertex $v$ are removed only when the improvement is substantial; namely, $d_G(v) > 2\mathsf{LD}_G(T'_v)$.

We conclude with the main theorem.

▶ **Theorem 18.** *The total number of edges in $G(V, E)$ is $\mathcal{O}(\mathsf{LD}_G(T^*) \log\log n)$. Thus, Algorithm 2 is an $\mathcal{O}(\log\log n)$-approximation algorithm for both the MLD and the respective CMC problems.*

## 5 Conclusions and Future Research

We gave an $\mathcal{O}(\log\log n)$-approximation algorithm for the MLD problem that implies an approximation algorithm with the same quality to the CMC problem. The approximation algorithm consists of local improvement steps. While we have evidence that the *analysis of our algorithm* is tight, it is open whether *the approximation ratio* we achieved is tight or whether there exists a $o(\log\log n)$-approximation for the MLD and CMC problems.

Our algorithm uses the fact that the weights of the vertices in the MLD problem are the degrees of these vertices in an undirected graph with no parallel edges. It will be interesting to see if the same approach can be extended to other weights. It cannot be extended to general weights as there are examples of graphs with general edge weights in which the ratio of the total edge weight to the weighted connected cut is $\Omega(\log\log n)$.

To the best of our knowledge neither nontrivial approximation algorithms nor hardness of approximation results are known for non-monotone submodular function maximization subject to connectivity constraints. Any progress along these lines for either general or specific classes of graphs is bound to have numerous applications.

## References


1 Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.*, 36(3):493–513, 1988.

2 Uriel Feige, Nicole Immorlica, Vahab S. Mirrokni, and Hamid Nazerzadeh. A combinatorial allocation mechanism with penalties for banner advertising. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 169–178, 2008.

3 Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vis.*, 59(2):167–181, 2004.

4 Rajiv Gandhi, Mohammad Taghi Hajiaghayi, Guy Kortsarz, Manish Purohit, and Kanthi K. Sarpatwar. On maximum leaf trees and connections to connected maximum cut problems. *Inf. Process. Lett.*, 129:31–34, 2018.

5 MohammadTaghi Hajiaghayi, Guy Kortsarz, Robert MacDavid, Manish Purohit, and Kanthi K. Sarpatwar. Approximation algorithms for connected maximum cut and related problems. In Nikhil Bansal and Irene Finocchi, editors, *ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 693–704. Springer, 2015.

6 MohammadTaghi Hajiaghayi, Guy Kortsarz, Robert MacDavid, Manish Purohit, and Kanthi K. Sarpatwar. Approximation algorithms for connected maximum cut and related problems. *Theor. Comput. Sci.*, 814:74–85, 2020.

**7** Jason D. Hartline, Vahab S. Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 189–198, 2008.

**8** Bart M. P. Jansen. Kernelization for maximum leaf spanning tree with positive vertex weights. *J. Graph Algorithms Appl.*, 16(4):811–846, 2012.

**9** Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

**10** Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. *J. ACM*, 51(2):263–280, 2004.

**11** Jon Lee, Viswanath Nagarajan, and Xiangkun Shen. Max-cut under graph constraints. In *Integer Programming and Combinatorial Optimization - 18th International Conference, IPCO, Liège, Belgium, June 1-3, 2016, Proceedings*, pages 50–62, 2016.

**12** Paul Lemke. The maximum leaf spanning tree problem for cubic graphs is NP-Complete. Technical report, University of Minnesota, 1988. Technical report, IMA publication no. 428.

**13** Slav Petrov. Image segmentation with maximum cuts, 2005.

**14** Alexander Reich. Complexity of the maximum leaf spanning tree problem on planar and regular graphs. *Theoretical Computer Science*, 626:134–143, 2016.

**15** Andreas S. Schulz and Nelson A. Uhan. Approximating the least core value and least core of cooperative games with supermodular costs. *Discrete Optimization*, 10(2):163–180, 2013.

**16** Roberto Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, volume 1461 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 1998.

# Parameterized Matroid-Constrained Maximum Coverage

**François Sellier** ✉
Université Paris Cité, CNRS, IRIF, F-75013, Paris, France
MINES ParisTech, Université PSL, F-75006, Paris, France

## Abstract

In this paper, we introduce the concept of *Density-Balanced Subset* in a matroid, in which independent sets can be sampled so as to guarantee that (i) each element has the same probability to be sampled, and (ii) those events are negatively correlated. These *Density-Balanced Subsets* are subsets in the ground set of a matroid in which the traditional notion of uniform random sampling can be extended.

We then provide an application of this concept to the *Matroid-Constrained Maximum Coverage* problem. In this problem, given a matroid $\mathcal{M} = (V, \mathcal{I})$ of rank $k$ on a ground set $V$ and a coverage function $f$ on $V$, the goal is to find an independent set $S \in \mathcal{I}$ maximizing $f(S)$. This problem is an important special case of the much-studied submodular function maximization problem subject to a matroid constraint; this is also a generalization of the maximum $k$-cover problem in a graph. In this paper, assuming that the coverage function has a bounded frequency $\mu$ (i.e., any element of the underlying universe of the coverage function appears in at most $\mu$ sets), we design a procedure, parameterized by some integer $\rho$, to extract in polynomial time an approximate kernel of size $\rho \cdot k$ that is guaranteed to contain a $1 - (\mu - 1)/\rho$ approximation of the optimal solution. This procedure can then be used to get a Fixed-Parameter Tractable Approximation Scheme (FPT-AS) providing a $1 - \varepsilon$ approximation in time $(\mu/\varepsilon)^{O(k)} \cdot |V|^{O(1)}$. This generalizes and improves the results of [Manurangsi, 2019] and [Huang and Sellier, 2022], providing the first FPT-AS working on an arbitrary matroid. Moreover, as the kernel has a very simple characterization, it can be constructed in the streaming setting.

**2012 ACM Subject Classification** Theory of computation → Packing and covering problems; Theory of computation → Fixed parameter tractability

**Keywords and phrases** Matroids, approximate kernel, maximum coverage

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.94

**Related Version** *Full Version*: `https://arxiv.org/abs/2308.06520`

**Funding** This work was funded by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

**Acknowledgements** The author thanks Chien-Chung Huang, Claire Mathieu, Eli Upfal, and the anonymous reviewers for their helpful comments.

## 1 Introduction

Matroids are fundamental combinatorial structures that generalize the notion of linear independence in a vector space as well as the notion of forest in a graph. In combinatorial optimization, the matroid constraints are an important generalization of the cardinality constraint. For instance, consider the problem of maximizing a submodular function under some constraint. If the constraint is that the feasible subsets are those of size bounded by some parameter $k$ (cardinality constraint), an approximation of $1 - 1/e$ can be obtained in polynomial time by a simple greedy algorithm [22] (this ratio is also the best possible in polynomial time unless $P = NP$, see [7]). Under the more general constraint that the feasible subsets are those that are independent in a given matroid $\mathcal{M}$ (matroid constraint), an approximation of $1 - 1/e$ can also be achieved in polynomial time [3], albeit by using


a more involved continuous greedy technique. Hence, somehow surprisingly, even though the matroid constraint is more complex than the cardinality constraint, some optimization problems are not really "harder" in the matroid context.

Following this perspective, in this paper we consider the problem of maximizing a coverage function of bounded frequency under some constraint. The starting point of our paper is the work of Manurangsi [18] in which, for cardinality constraints, a Fixed-Parameter Tractable Approximation Scheme (FPT-AS) is developed. Our main result here is a generalization of that FPT-AS to matroid constraints (Corollary 8), extending the approximate kernel construction of [18] to matroids (Theorem 6). A key idea in [18] is the use of uniform random sampling of subsets of given cardinality; unfortunately, in the matroid setting, near-uniform sampling of independent sets is in general impossible. Instead, here we introduce the concept of *Density-Balanced Subset* (DBS, Definition 2) in matroids. We show that in those particular subsets of the ground set we can generalize the traditional notion of uniform random sampling of a subset of a given cardinality, to that of sampling a maximum independent set, while guaranteeing that (i) every element of the DBS has the same probability of being sampled (i.e., the probabilities are "balanced") and that (ii) those events are negatively correlated (Proposition 3).

## Density-Balanced Subsets

We introduce here the concept of *Density-Balanced Subsets*. Let us first define the notion of density. (In the following we assume that readers already have some familiarity with matroids; an introduction to matroids is provided in the beginning of Section 2.)

▶ **Definition 1.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid. The* density *of a subset $U \subseteq V$ in $\mathcal{M}$ is defined as*

$$\rho_{\mathcal{M}}(U) = \frac{|U|}{\mathrm{rank}_{\mathcal{M}}(U)}.$$

*The density of an empty set is set to 0, and the density of a non-empty set of rank 0 is $+\infty$.*

From that we define a *Density-Balanced Subset* (DBS). Basically, in a DBS, no subset has a larger density than the DBS itself.

▶ **Definition 2.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and $\rho$ be a positive integer. A subset $V' \subseteq V$ is called a* $\rho$-DBS *in $\mathcal{M}$ if $\rho_{\mathcal{M}}(V') = \rho$ and for all $U \subseteq V'$, $\rho_{\mathcal{M}}(U) \leq \rho$.*

Density-Balanced Subsets appear naturally when extracting independent sets in matroid unions (as we will see in Section 3); in Figure 1 a simple example of DBS is given.

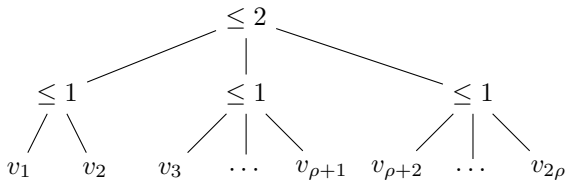



**Figure 1** Example of $\rho$-DBS of rank $k = 2$. The tree represents a laminar matroid $\mathcal{M} = (V, \mathcal{I})$ on the ground set $V = \{v_1, \ldots, v_{2\rho}\}$: the leaves represent elements of the ground set, and the inner nodes represent cardinality constraints on the elements in their associated subtree (e.g., if $S \in \mathcal{I}$, then $|S \cap \{v_3, \ldots, v_{\rho+1}\}| \leq 1$). Observe that $V' = V$ is a $\rho$-DBS.

In $\rho$-DBSes, it is possible to sample independent sets while having the desired balance and negative correlation properties. Moreover, the sampled independent sets are "maximal" (i.e., these sampled independent sets are bases in the restriction of the matroid to the DBS), hence, in this sense, this extends the notion of uniform random sampling. This result comes from a more general randomized rounding algorithm developed by Chekuri, Vondrák, and Zenklusen [4].

▶ **Proposition 3.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, $V' \subseteq V$ be a $\rho$-DBS for some positive integer $\rho$, and $k = \operatorname{rank}_{\mathcal{M}}(V')$. There exists a procedure to sample randomly from $V'$ an independent set of $k$ elements $S = \{s_1, \ldots, s_k\} \in \mathcal{I}$ such that:*

**(i)** *for all $v \in V'$, $\mathbb{P}[v \in S] = 1/\rho$;*

**(ii)** *for all $T \subseteq V'$, $\mathbb{P}[T \subseteq S] \leq \prod_{v \in T} \mathbb{P}[v \in S] = 1/\rho^{|T|}$.*

## Matroid-Constrained Maximum Coverage

Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid of rank $k$ on a ground set $V$. Given a universe $U$, a weight function $w : U \to \mathbb{R}_+$, and a family $\{U_v\}_{v \in V}$ of subsets of $U$, the *matroid-constrained maximum coverage* problem is to select a subset $S \in \mathcal{I}$ maximizing the coverage function $f(S) = w(\bigcup_{s \in S} U_s) = \sum_{u \in \bigcup_{s \in S} U_s} w(u)$, namely, to find an element in $\arg\max_{S \in \mathcal{I}} f(S)$.

The *frequency* of an element of the universe $u \in U$ refers to the number of sets $U_v$ in which it appears. We say that $f$ has *bounded frequency* $\mu$ if every element of the underlying universe $U$ has a frequency bounded by $\mu$. In our paper, we will focus on the matroid-constrained maximum coverage problem for coverage functions having some bounded frequency $\mu$; this assumption is quite common and is used for instance in [2, 8, 13, 21, 24, 25]. An important special case is $\mu = 2$, where it corresponds to the coverage function over edges in a graph. Besides the frequency parameter, another parameter $z$, corresponding to the number of points covered in an optimal solution, has been used to design FTP algorithms in [15].

To put our problem in a larger picture, we can first observe that a coverage function is a special case of monotone submodular function. For the problem of maximizing a monotone submodular function under a matroid constraint, an approximation of $1 - 1/e$ can be achieved in polynomial time [3] using continuous greedy and pipage rounding techniques. Later, a combinatorial approach for maximizing coverage functions over matroids was developed to achieve the same ratio [9], and this approach was then generalized to monotone submodular functions [10].

A special case of our problem when the matroid constraint is simply a cardinality constraint (i.e., a uniform matroid) has been studied in [1, 7, 12, 26]. It has been shown in [12] that a simple greedy procedure (picking at each step the element maximizing the increase of the coverage function) guarantees a ratio of $1 - 1/e$. If a polynomial time algorithm could approximate maximum coverage within a ratio of $1 - 1/e + \varepsilon$ for some $\varepsilon > 0$, then it would imply that $P = NP$ [7]. Furthermore, one cannot obtain in FPT time (where the matroid rank $k$ is the parameter) an approximation ratio better than $1 - 1/e + \varepsilon$, assuming GAP-ETH [19]. However, when the coverage function has bounded frequency $\mu$, an approximation of $1 - (1 - 1/\mu)^{\mu}$ can be achieved [1].

The case where the coverage function has a frequency $\mu$ bounded by 2 is called the *matroid-constrained vertex cover* [13, 14], and is called MAX $k$-VERTEX-COVER when the matroid is uniform. The latter has also been studied through the lens of fixed-parameterized-tractability. The problem is $W[1]$-hard with $k$ being the parameter [11], thus getting an exact solution in FPT time is unlikely. Nonetheless, it is possible to get a near-optimal solution in FPT time [20]. Precisely, an FPT approximation scheme (FPT-AS) is given in [20], that delivers a $1 - \varepsilon$ approximate solution in $(k/\varepsilon)^{O(k^3/\varepsilon)} \cdot |V|^{O(1)}$ time, later improved to $(1/\varepsilon)^{O(k)} \cdot |V|^{O(1)}$ in [18, 25].

Here we recall the definition of an FPT-AS, introduced by Marx [20]:

▶ **Definition 4.** *Given a parameter function $\kappa$ associating a positive integer to each instance $x \in I$ of some problem, a* Fixed-Parameter Tractable Approximation Scheme (FPT-AS) *is an algorithm that provides a $(1-\varepsilon)$ approximate solution in time $g(\varepsilon, \kappa(x)) \cdot |x|^{O(1)}$ for some computable function $g$.*

In our case, each of the instances consists of a bounded-frequency coverage function and a matroid, and the parameter of an instance is the rank $k$ of its matroid.

For our problem, an FPT-AS has been designed for partition, laminar, and transversal matroids in [13], where the concept of *robust subset* is introduced to generalize the random sampling argument developed in [18]. In [13], an approximate kernel[1] is extracted, consisting of a maximum weight independent set in the union of several copies of the same matroid $\mathcal{M}$ (see below for a definition of the union of matroids), and then a brute-force enumeration is performed on that kernel of small size. The number of matroids in these unions depends on the type of matroid.

▶ **Definition 5.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid. Then we can define $\rho\mathcal{M} = (V, \mathcal{I}_\rho)$ as the* union of $\rho$ matroids $\mathcal{M}$*, as follows: $S \in \mathcal{I}_\rho$ if $S$ can be partitioned into $S_1 \cup \cdots \cup S_\rho$ so that for all $i$ we have $S_i \in \mathcal{I}$.*

It is known that the union of matroids is still a matroid and that an independence oracle for $\rho\mathcal{M}$ can be implemented in polynomial time given an independence oracle for $\mathcal{M}$, e.g., see [23]. Moreover, the rank of $\rho\mathcal{M}$ is at most $\rho$ times the rank of $\mathcal{M}$.

In this paper, we prove that a maximum weight independent set in $\rho\mathcal{M}$ contains an approximate solution of the problem, and that this ratio does not depend on the type of matroid (unlike [13]):

▶ **Theorem 6.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid and let $f$ be a coverage function on $V$ of frequency bounded by $\mu$. Let $V'$ be a maximum weight independent set in $\rho\mathcal{M}$, with respect to the weights $f(\{v\})$. Then $V'$ contains a $1 - (\mu - 1)/\rho$ approximate solution of the matroid-constrained maximum coverage problem.*

The proof of Theorem 6 relies on a reinterpretation of the greedy algorithm (extracting $V'$ in the matroid $\rho\mathcal{M}$, i.e., Algorithm 2) as the process of constructing $\rho$-DBSes in a series of contracted matroids of $\mathcal{M}$ (see Algorithm 1). These DBSes that appear during the construction of the kernel can then be used for random sampling purposes, allowing us to generalize the argument of [18] for uniform matroids to any matroid (details in Section 3).

▶ Remark 7. The simplicity of our characterization of the kernel implies that our kernelization process can be easily turned into a streaming algorithm, as in [13]. In fact, assuming that the matroid as well as the cover function is provided to the algorithm as oracles, maintaining a maximum weight independent set in $\rho\mathcal{M}$ with respect to the weights $f(\{v\})$ can be done in streaming using $O(\rho \cdot k)$ memory.

By taking the appropriate value $\rho = \lceil (\mu - 1)/\varepsilon \rceil$ and performing a bruteforce enumeration on the approximate kernel described in Theorem 6 (that kernel would be of size $\rho \cdot k = O(k \cdot \mu/\varepsilon)$ and could be extracted in polynomial time, assuming that an independence oracle for $\mathcal{M}$ and an oracle for $f$ are given), we obtain an FPT-AS, extending the result of [18]:

[1] An $\alpha$-approximate kernel [17, 18] for some parameterized optimization problem is a pair of polynomial time algorithms $\mathcal{A}$, the *reduction algorithm*, and $\mathcal{B}$, the *solution lifting algorithm*, such that (i) given an instance $(x, \kappa(x))$, $\mathcal{A}$ produces another instance $(x', \kappa(x'))$ such that $|x'|$, $\kappa(x')$ are bounded by $g(\kappa(x))$ and (ii) given a $\beta$ approximate solution $S'$ for $(x', \kappa(x'))$, $\mathcal{B}$ produces a solution $S$ of $(x, \kappa(x))$ such that $S$ is an $\alpha\beta$ approximate solution for $(x, \kappa(x))$. In the following we will use the terms "kernel" and "approximate kernel" interchangeably, dropping the adjective.

▶ **Corollary 8.** *There exists an algorithm that computes a $1-\varepsilon$ approximate solution of the matroid-constrained maximum coverage in $(\mu/\varepsilon)^{O(k)} \cdot |V|^{O(1)}$ time.*

To see the interest of this result, we note that if the only parameter is the rank $k$ of the matroid, one cannot obtain in FPT time an approximation ratio better than $1-1/e+\varepsilon$ (assuming GAP-ETH) for a general coverage function [19], even if the matroid is the simplest uniform matroid. In contrast, our result shows that it is possible to break through this lower bound and get arbitrarily close to 1 for an arbitrary matroid when the frequency of the coverage function is bounded. We also emphasize here that the randomization is only used in the analysis; our algorithm itself is deterministic.

An important special case of Theorem 6 is when $\mu = 2$, which corresponds to the matroid-constrained vertex cover problem, and for which we have:

▶ **Corollary 9.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid and let $G = (V, E)$ be a weighted graph. Let $V'$ be a maximum weight independent set in $\rho\mathcal{M}$, with respect to the weighted degrees $\deg_w(v)$. Then $V'$ contains a $1-1/\rho$ approximate solution of the matroid-constrained vertex cover problem.*

This extends and improves the previous kernelization results for this problem [13, 18]. In fact, in [18] the $1-1/\rho$ approximation is attained for the union of $\rho$ uniform matroids, and in [13] that ratio is attained either for the union of $\rho$ partition matroids, $2 \cdot \rho$ laminar matroids, or $\rho + k - 1$ (reduced later to $\rho$ in [16]) transversal matroids.

## 2 Density-Balanced Subsets

Let us start with some definitions. Given a finite set $V$, a matroid is a pair $\mathcal{M} = (V, \mathcal{I})$ where $\mathcal{I} \subseteq \mathcal{P}(V)$ is a family of subsets in $V$ that satisfies the following three conditions: (1) $\emptyset \in \mathcal{I}$, (2) if $X \subseteq Y \in \mathcal{I}$, then $X \in \mathcal{I}$, and (3) if $X, Y \in \mathcal{I}, |Y| > |X|$, then there exists an element $e \in Y \backslash X$ so that $X \cup \{e\} \in \mathcal{I}$.

The set $V$ is called the *ground set* of the matroid and the elements of $\mathcal{I}$ are called the *independent sets*. Matroids terminology borrows concepts from vector spaces as well as graph theory. The *rank* of a subset $X \subseteq V$ is defined as $\text{rank}_{\mathcal{M}}(X) = \max_{Y \subseteq X, Y \in \mathcal{I}} |Y|$; the rank of a matroid is defined as $\text{rank}_{\mathcal{M}}(V)$. The *span* of a subset $X \subseteq V$ in the matroid $\mathcal{M}$ is defined as $\text{span}_{\mathcal{M}}(X) = \{x \in V : \text{rank}_{\mathcal{M}}(X \cup \{x\}) = \text{rank}_{\mathcal{M}}(X)\}$, and these elements in the *span* are called *spanned by* $X$ in $\mathcal{M}$. A subset $C \subseteq V$ is a *circuit* if $C$ is a minimal non-independent set, i.e., for every $v \in C$, $C \backslash \{v\} \in \mathcal{I}$. An element in $V$ that is a circuit by itself is called a *loop*. For more details about matroids, we refer the reader to [23].

We recall the definition of a *restriction* and a *contraction* of a matroid. Performing such operation on a matroid results in another matroid.

▶ **Definition 10** (Restriction). *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and let $V' \subseteq V$ be a subset. Then we define the* restriction of $\mathcal{M}$ to $V'$ *as $\mathcal{M}|V' = (V', \mathcal{I}')$ where $\mathcal{I}' = \{S \subseteq V' : S \in \mathcal{I}\}$*

▶ **Definition 11** (Contraction). *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and let $U$ be a subset of $V$. Then we define the* contracted matroid *$\mathcal{M}/U = (V \backslash U, \mathcal{I}_U)$ so that, given a maximum independent subset $\mathcal{B}_U$ of $U$, $\mathcal{I}_U = \{S \subseteq V \backslash U : S \cup \mathcal{B}_U \in \mathcal{I}\}$.*

It is well-known that any choice of $\mathcal{B}_U$ produces the same $\mathcal{I}_U$, as a result the definition of contraction is unambiguous. The following proposition comes directly from the definition.

▶ **Proposition 12.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid and let $A \subseteq B \subseteq V$. Then we have $\text{rank}_{\mathcal{M}/A}(B \backslash A) = \text{rank}_{\mathcal{M}}(B) - \text{rank}_{\mathcal{M}}(A)$.*

In this paper, we use the notion of density of a subset in a matroid. We can observe from that definition that the density of a non-empty set is always larger or equal to one.

▶ **Definition 1.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid. The* density *of a subset $U \subseteq V$ in $\mathcal{M}$ is defined as*

$$\rho_{\mathcal{M}}(U) = \frac{|U|}{\mathrm{rank}_{\mathcal{M}}(U)}.$$

*The density of an empty set is set to 0, and the density of a non-empty set of rank 0 is $+\infty$.*

We now introduce the notion of *Density-Balanced Subset* (DBS).

▶ **Definition 2.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and $\rho$ be a positive integer. A subset $V' \subseteq V$ is called a* $\rho$-DBS *in $\mathcal{M}$ if $\rho_{\mathcal{M}}(V') = \rho$ and for all $U \subseteq V'$, $\rho_{\mathcal{M}}(U) \leq \rho$.*

Here is a theorem due to Edmonds [5] that will allow us to get another characterization of Density-Balanced Subsets.

▶ **Theorem 13** (Theorem 1 in [5])**.** *The elements of a matroid $\mathcal{M}$ can be partitioned into as few as $\rho$ sets, each of which is independent, if and only if there is no subset $A$ of elements of $\mathcal{M}$ for which $|A| > \rho \cdot \mathrm{rank}_{\mathcal{M}}(A)$.*

▶ **Proposition 14.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid. If $V'$ is a $\rho$-DBS for some positive integer $\rho$, then there exist $\rho$ independent sets $B_1, \ldots, B_\rho$ such that $V' = B_1 \cup \cdots \cup B_\rho$. Conversely, if a set $V'$ of density $\rho$ can be partitioned into $\rho$ independent sets $B_1, \ldots, B_\rho$, then $V'$ is a $\rho$-DBS.*

**Proof.** Consider the matroid $\mathcal{M}|V'$, i.e., the restriction of $\mathcal{M}$ to $V'$. By Theorem 13, as the density of any set $A \subseteq V'$ is bounded by $\rho$, $V'$ can be partitioned into $\rho$ independent sets $B_1, \ldots, B_\rho$. Conversely, if $V'$ can be partitioned into $\rho$ independent sets $B_1, \ldots, B_\rho$, then by Theorem 13 the density of any set $A \subseteq V'$ is bounded by $\rho$. As we assumed $\rho_{\mathcal{M}}(V') = \rho$, $V'$ is a $\rho$-DBS. ◀

In DBSes, the notion of uniform random sampling can be properly extended.

▶ **Proposition 3.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, $V' \subseteq V$ be a $\rho$-DBS for some positive integer $\rho$, and $k = \mathrm{rank}_{\mathcal{M}}(V')$. There exists a procedure to sample randomly from $V'$ an independent set of $k$ elements $S = \{s_1, \ldots, s_k\} \in \mathcal{I}$ such that:*

**(i)** *for all $v \in V'$, $\mathbb{P}[v \in S] = 1/\rho$;*

**(ii)** *for all $T \subseteq V'$, $\mathbb{P}[T \subseteq S] \leq \prod_{v \in T} \mathbb{P}[v \in S] = 1/\rho^{|T|}$.*

**Proof.** By Proposition 14, we can write $V' = B_1 \cup \cdots \cup B_\rho$ for some disjoint independent sets. Hence we obtain (denoting $\mathbb{1}_U$ the indicator vector of the set $U$):

$$\frac{1}{\rho}\mathbb{1}_{V'} = \frac{1}{\rho}\sum_{i=1}^{\rho} \mathbb{1}_{B_i} \in P(\mathcal{M}),$$

where $P(\mathcal{M}) = \mathrm{conv}\{\mathbb{1}_S : S \in \mathcal{I}\} = \{x \in [0,1]^V : \forall\, S \subseteq V, \sum_{v \in S} x_v \leq \mathrm{rank}_{\mathcal{M}}(S)\}$ denotes the matroid polytope of $\mathcal{M}$. Therefore we can apply the randomized rounding algorithm developed in [4] to the vector $\frac{1}{\rho}\mathbb{1}_{V'}$ to get an integral vector $X = \mathbb{1}_S \in \{0,1\}^V$ such that $S \subseteq V'$ is a maximum independent set and

**(i)** for all $v \in V'$, $\mathbb{P}[v \in S] = \mathbb{E}[X_v] = 1/\rho$;

**(ii)** for all $T \subseteq V'$, $\mathbb{P}[T \subseteq S] = \mathbb{E}[\prod_{v \in T} X_v] \leq \prod_{v \in X} \mathbb{E}[X_v] = \prod_{v \in T} \mathbb{P}[v \in S] = 1/\rho^{|T|}$.

This concludes the proof. ◀

Conversely we also have the following proposition.

▶ **Proposition 15.** *Let $V'$ a set of integer density $\rho$ in which independent sets can be sampled randomly so that each element has probability $1/\rho$ of being sampled, then $V'$ is a $\rho$-DBS.*

**Proof.** In fact, if there exists $U \subseteq V'$ such that $\rho_{\mathcal{M}}(U) > \rho$, then an algorithm sampling each element in $V'$ with probability $1/\rho$ would take in expectation strictly more than $\mathrm{rank}_{\mathcal{M}}(U)$ elements in $U$, meaning that some of those sampled set violate the rank constraint on $U$. ◀

Another useful property of DBSes is that after a matroid contraction we can still recover in a $\rho$-DBS $V'$ a smaller $V''$ while preserving the rank of $V'$ in the contracted matroid:

▶ **Proposition 16.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and $V'$ be a $\rho$-DBS in $\mathcal{M}$ for some positive integer $\rho$. Let $A \subseteq V$ such that $V' \not\subseteq \mathrm{span}_{\mathcal{M}}(A)$. Then there exists a subset $V'' \subseteq V'$ such that:*

**(i)** *$V''$ is a $\rho$-DBS in $\mathcal{M}/A$;*

**(ii)** $\mathrm{rank}_{\mathcal{M}/A}(V'') = \mathrm{rank}_{\mathcal{M}/A}(V' \backslash A)$.

**Proof.** Using Proposition 14, we know that $V' = B_1 \cup \cdots \cup B_\rho$ for some disjoint independent sets $B_1, \ldots, B_\rho$, each of cardinality $\mathrm{rank}_{\mathcal{M}}(V')$ (because $V'$ has density $\rho$, so it contains $\rho \cdot \mathrm{rank}_{\mathcal{M}}(V')$ elements, and each independent set is made of at most $\mathrm{rank}_{\mathcal{M}}(V')$ elements), and each of them spanning $V'$ in $\mathcal{M}$. As a result, for all $i \in \{1, \ldots, \rho\}$ we also have $V' \backslash A \subseteq \mathrm{span}_{\mathcal{M}/A}(B_i \backslash A)$, hence $\mathrm{rank}_{\mathcal{M}/A}(B_i \backslash A) = \mathrm{rank}_{\mathcal{M}/A}(V' \backslash A)$ and thereby there exists $B'_i \subseteq B_i \backslash A$ such that $B'_i$ is independent in $\mathcal{M}/A$ and $|B'_i| = \mathrm{rank}_{\mathcal{M}/A}(V' \backslash A)$. Hence $V'' = B'_1 \cup \cdots \cup B'_\rho$ is a set of rank equal to $\mathrm{rank}_{\mathcal{M}/A}(V' \backslash A)$ in $\mathcal{M}/A$, contains $\rho \cdot \mathrm{rank}_{\mathcal{M}/A}(V' \backslash A)$ elements, and can be partitioned into $\rho$ independent sets, therefore, by Proposition 14, $V''$ is a $\rho$-DBS in $\mathcal{M}/A$ with $\mathrm{rank}_{\mathcal{M}/A}(V'') = |B'_1| = \cdots = |B'_\rho| = \mathrm{rank}_{\mathcal{M}/A}(V' \backslash A)$. ◀

The following propositions are about matroid contraction and densest subsets. The next proposition states how the density is changed after a matroid contraction.

▶ **Proposition 17.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid. If $A \subseteq B \subseteq V$ and $U \subseteq V \backslash B$ we have $\rho_{\mathcal{M}/A}(U) \leq \rho_{\mathcal{M}/B}(U)$, assuming that $\rho_{\mathcal{M}/A}(U) < +\infty$.*

**Proof.** We have $\mathrm{rank}_{\mathcal{M}/A}(U) \geq \mathrm{rank}_{\mathcal{M}/B}(U)$, while the cardinality $|U|$ remains the same. ◀

Now we give some results regarding densest subsets, which are closely related to *Density-Balanced Subsets*, as a densest subset in a matroid is automatically a DBS.

▶ **Proposition 18.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, and $\rho$ be a positive integer. Let $V' \subseteq V$. If $\max_{U \subseteq V'} \rho_{\mathcal{M}}(U) < \rho$, then for any $v \in V \backslash V'$, $\max_{U \subseteq V' \cup \{v\}} \rho_{\mathcal{M}}(U) \leq \rho$.*

**Proof.** Consider $U \subseteq V'$, $U \neq \emptyset$. As $\rho_{\mathcal{M}}(U) < \rho$, we know that $|U| \leq \rho \cdot \mathrm{rank}_{\mathcal{M}}(U) - 1$ (because $\rho \cdot \mathrm{rank}_{\mathcal{M}}(U)$ is an integer). Therefore we have

$$\rho_{\mathcal{M}}(U \cup \{v\}) \leq \frac{|U| + 1}{\mathrm{rank}_{\mathcal{M}}(U)} \leq \frac{\rho \cdot \mathrm{rank}_{\mathcal{M}}(U) - 1 + 1}{\mathrm{rank}_{\mathcal{M}}(U)} = \rho,$$

which concludes the proof. ◀

▶ **Proposition 19.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid, $V'$ be a subset of $V$, and let $B$ be a subset that reaches the maximum density $\rho^* < +\infty$ in $V'$. Then given any $A \subsetneq B$, $\rho_{\mathcal{M}/A}(B \backslash A) \geq \rho^*$.*

**Proof.** If $\text{rank}_{\mathcal{M}/A}(B\backslash A) = 0$ then $\rho_{\mathcal{M}/A}(B\backslash A) = +\infty$ and we are done; otherwise, by Proposition 12:

$$\rho_{\mathcal{M}}(B) = \frac{\text{rank}_{\mathcal{M}}(A)\cdot\rho_{\mathcal{M}}(A) + \text{rank}_{\mathcal{M}/A}(B\backslash A)\cdot\rho_{\mathcal{M}/A}(B\backslash A)}{\text{rank}_{\mathcal{M}}(A) + \text{rank}_{\mathcal{M}/A}(B\backslash A)},$$

hence $\rho_{\mathcal{M}}(B)$ is a weighted average of $\rho_{\mathcal{M}}(A)$ and $\rho_{\mathcal{M}/A}(B\backslash A)$. As $\rho_{\mathcal{M}}(A) \leq \rho^*$ (by definition of $\rho^*$), it implies that $\rho_{\mathcal{M}/A}(B\backslash A) \geq \rho^*$. ◀

The following proposition states that the densest subsets are closed under union, thus proving the uniqueness of the maximum cardinality densest subset – that property will be useful in Algorithm 1 (Section 3).

▶ **Proposition 20.** *Let $\mathcal{M} = (V,\mathcal{I})$ be a matroid, and $V' \subseteq V$. Let $\rho^* = \max_{U\subseteq V'}\rho_{\mathcal{M}}(U) < +\infty$. Then given any two subsets $W_1$, $W_2$ in $V'$ of density $\rho^*$, $\rho_{\mathcal{M}}(W_1 \cup W_2) = \rho^*$.*

**Proof.** If $W_1 \subseteq W_2$, then the proposition is trivially true. So assume that $W_1\backslash W_2 \neq \emptyset$, and we can observe that

$$\rho^* \leq \rho_{\mathcal{M}/(W_1\cap W_2)}(W_1\backslash(W_1\cap W_2)) \leq \rho_{\mathcal{M}/W_2}(W_1\backslash(W_1\cap W_2)),$$

where the first inequality uses Proposition 19 and the second uses Proposition 17. As a result, by the facts that $\rho_{\mathcal{M}}(W_2) = \rho^*$ and that $\rho_{\mathcal{M}/W_2}(W_1\backslash(W_1\cap W_2)) \geq \rho^*$, we obtain $\rho_{\mathcal{M}}(W_1\cup W_2) \geq \rho^*$ (using a weighted average argument as in Proposition 19). Hence we have $\rho_{\mathcal{M}}(W_1\cup W_2) = \rho^*$. ◀

▶ **Proposition 21.** *Let $\mathcal{M} = (V,\mathcal{I})$ be a matroid, and $V' \subseteq V$ a non-empty set. Let $A$ be the largest densest subset in $V'$. Then for any $B \subseteq V'\backslash A$, we have $\rho_{\mathcal{M}/A}(B) < \rho_{\mathcal{M}}(A)$.*

**Proof.** We proceed by contradiction. Suppose that there exists $B \subseteq V'\backslash A$ such that $\rho_{\mathcal{M}/A}(B) \geq \rho_{\mathcal{M}}(A)$. Then it implies that

$$\rho_{\mathcal{M}}(A\cup B) = \frac{\rho_{\mathcal{M}}(A)\cdot\text{rank}_{\mathcal{M}}(A)}{\text{rank}_{\mathcal{M}}(A)+\text{rank}_{\mathcal{M}/A}(B)} + \frac{\rho_{\mathcal{M}/A}(B)\cdot\text{rank}_{\mathcal{M}/A}(B)}{\text{rank}_{\mathcal{M}}(A)+\text{rank}_{\mathcal{M}/A}(B)} \geq \rho_{\mathcal{M}}(A),$$

contradicting the hypothesis that $A$ was the largest densest set in $V'$. ◀

## 3 Matroid-Constrained Maximum Coverage

Here we will use a slightly different formalization of the coverage function, based on edge-weighted hypergraphs (compared to the one given in the introduction), but it is straightforward to see that these formalizations are equivalent. Let $G = (V,E)$ be a hypergraph, $V$ being a set of vertices and $E$ being a set of hyper-edges, i.e., an element $e \in E$ is a subset of $V$. We denote $n = |V|$. Let $w : E \to \mathbb{R}_+$ be a weight function on the hyper-edges. We extend this function to any set of hyper-edges by setting for any $A \subseteq E$, $w(A) = \sum_{e\in A} w(e)$. For a vertex $v \in V$, we denote $\delta(v)$ the set of its set of *incident* hyper-edges, namely, $\delta(v) = \{e \in E : v \in e\}$, and $\deg_w(v)$ its *weighted degree*, namely, $\deg_w(v) = w(\delta(v))$. The *frequency* of a hyper-edge $e$ is defined as the number of vertices for which $e$ appears in $\delta(v)$, namely, $|e|$. The hypergraph $G$ will be said of *bounded frequency* $\mu$ if all its hyper-edges have frequencies bounded by $\mu$. For two sets of vertices $S$, $T$ we denote by $E(S,T)$ the set of hyper-edges having at least one endpoint in each set $S$ and $T$, namely, $E(S,T) = \{e \in E : e\cap S \neq \emptyset, e\cap T \neq \emptyset\}$. For conciseness we will denote $E(S) = E(S,S)$.

Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid on the ground set $V$. In the matroid-constrained maximum coverage problem, we are asked to find a set of vertices $S \subseteq V$ that is independent in the matroid $\mathcal{M}$ (i.e., $S \in \mathcal{I}$) and that maximizes the total weight of the covered hyperedges, namely, an element of $\arg\max_{S \in \mathcal{I}} w(E(S))$. The problem can be solved exactly by the standard greedy algorithm if $\mu = 1$ [6], so in the following we will assume that our hypergraph has a bounded frequency of $\mu \geq 2$. Observe that the case $\mu = 2$ corresponds to the matroid-constrained vertex cover, studied in [13].

Here we want to construct a kernel that contains a good approximation of the optimal solution of the maximum coverage problem under a matroid constraint. We will start by describing a procedure to build the kernel that will be convenient for the analysis, and then we will show that this algorithm turn out to be equivalent to the one of Theorem 6. We build our kernel $V'$ as follows. Let $\rho$ be a fixed positive integer. Start with an empty set $V'$, and an auxiliary set $C$ that is also empty at the beginning. Process the elements $v_i \in V$ in the order of non-increasing weighted-degrees. If the element $v_i$ is not spanned by $V'$ at that time we add that element to $C$, we check whether $C$ contains a set of density larger or equal to $\rho$ with respect to the matroid $\mathcal{M}/V'$. If this is the case, then we consider that largest densest subset $X$ in $C$ with respect to $\mathcal{M}/V'$, we add that set into $V'$ and remove that set from $C$ (note that, because of Proposition 20, the largest densest set $X$ is well-defined). When the main loop terminates, the set $C$ is also added into $V'$. A formal description of this procedure is provided in Algorithm 1.

**Algorithm 1** Algorithm for building a maximum coverage approximate kernel.

```
1: V = {v_1, ..., v_n} where deg_w(v_1) ≥ ··· ≥ deg_w(v_n)
2: V' ← ∅, C ← ∅
3: for i = 1, ..., n do  ▷ vertices are processed in non-increasing order of weighted degree
4:     if v_i ∈ span_M(V') then
5:         continue                          ▷ v_i is ignored if already spanned by V'
6:     C ← C ∪ {v_i}
7:     let X be the largest densest subset in C with respect to the matroid M/V'
8:     if ρ_{M/V'}(X) ≥ ρ then
9:         C ← C\X
10:        V' ← V' ∪ X
11: V' ← V' ∪ C
12: return V'
```

▷ Claim 22. In Algorithm 1, at the end of each iteration of the loop, for all $U \subseteq C$, $\rho_{\mathcal{M}/V'}(U) < \rho$. Moreover, when the condition at Line 8 is true, we have $\rho_{\mathcal{M}/V'}(X) = \rho$.

Proof. We prove these two properties by induction. In the first iteration of the algorithm, both properties are clearly true. Suppose that during the $i$th iteration both properties are satisfied. It means that at the beginning of the $(i+1)$st iteration the densest subset in $C$ is of density strictly smaller than $\rho$, hence Proposition 18 implies that the densest subset after inserting $v_{i+1}$ in $C$ cannot be of density strictly larger than $\rho$. This implies the second property for the $(i+1)$st iteration. Regarding the first property,

- if condition at Line 8 is false, then the first property is clearly satisfied;
- if condition at Line 8 is true, then the preceding discussion implies that the largest densest subset $X$ removed from $C$ has density $\rho$, and therefore, by Proposition 21, the first property is satisfied.

This concludes the proof. ◁

The set $V'$ can be decomposed as follows:

$$V' = X_1 \cup \cdots \cup X_r \cup R \tag{1}$$

where the $X_i$s represent the largest densest subsets $X$ that were added through the execution of the algorithm (at Line 10), labeled in the order they were added, and $R$ represents the set of remaining elements coming from $C$ that were added after termination of the main loop (at Line 11). As each $X_i$ is a densest subset of density $\rho$ in $\mathcal{M}/(\bigcup_{j=1}^{i-1} X_j)$, each set $X_i$ is a $\rho$-DBS in the matroid $\mathcal{M}/(\bigcup_{j=1}^{i-1} X_j)$.

Now we can prove the following important lemma, which states that the set $V'$ built by Algorithm 1 is an approximate kernel, i.e., a small subset of $V$ containing a good approximation of the optimal solution:

▶ **Lemma 23.** *Let $V'$ be the kernel built in Algorithm 1, and let $O$ be an optimal solution. Then $V'$ contains an independent set $S$ such that $w(E(S)) \geq (1 - (\mu - 1)/\rho) \cdot w(E(O))$.*

**Proof.** Let $O \in \mathcal{I}$ be an optimal solution. We denote $O^{in} = O \cap V'$, $O^{out} = O \backslash O^{in}$. As in [13, 18], we want to sample randomly an independent set $S \subseteq V'$ so that we have an inequality *in expectation*, implying that some set satisfying that same inequality actually exists:

$$\mathbb{E}[w(E(S))] \geq (1 - (\mu - 1)/\rho) \cdot w(E(O)).$$

To sample $S$, we will do the following. For $i = 1, \ldots, r$, using Proposition 16, consider $X'_i \subseteq X_i$ a $\rho$-DBS in $\mathcal{M}/(O^{in} \cup \bigcup_{j=1}^{i-1} X_j)$ of rank $k_i = \operatorname{rank}_{\mathcal{M}/(O^{in} \cup \bigcup_{j=1}^{i-1} X_j)}(X_i)$ (if $X_i$ is spanned by $O^{in} \cup \bigcup_{j=1}^{i-1} X_j$ in $\mathcal{M}$, then we just set $X'_i = \emptyset$ and $k_i = 0$). From that set $X'_i$, using Proposition 3, we sample an independent set $S_i = \{s_{i,1}, \ldots, s_{i,k_i}\} \subseteq X'_i$ in $\mathcal{M}/(O^{in} \cup \bigcup_{j=1}^{i-1} X_j)$ such that:

**(i)** for all $v \in X'_i$, $\mathbb{P}[v \in S_i] = 1/\rho$;
**(ii)** for all $v, v' \in X'_i$ such that $v \neq v'$, $\mathbb{P}[v \in S_i \wedge v' \in S_i] \leq 1/\rho^2$.

Then we set $S = O^{in} \cup \bigcup_{i=1}^{r} S_i$. We denote $\tilde{S} = \bigcup_{i=1}^{r} S_i$ and $V'' = \bigcup_{i=1}^{r} X'_i$. The $S_i$s are sampled independently.

▷ Claim 24. The set $S$ sampled using the aforementioned method is always independent in $\mathcal{M}$. Moreover,

**(i)** for all $v \in V''$, $\mathbb{P}[v \in \tilde{S}] = 1/\rho$;
**(ii)** for all $v, v' \in V''$ such that $v \neq v'$, $\mathbb{P}[v \in \tilde{S} \wedge v' \in \tilde{S}] \leq 1/\rho^2$.

Proof. We prove by induction on $i$ that $O^{in} \cup \bigcup_{j=1}^{i} S_j$ is independent in $\mathcal{M}$. For $i = 0$ this is clearly true, as $O^{in}$ is a subset of $O$, which is an independent set in $\mathcal{M}$. Suppose that the property is true for some $i < r$. Then it means that $O^{in} \cup \bigcup_{j=1}^{i} S_j$ is an independent set in $\mathcal{M}$. We know that for any sampling in $X'_{i+1}$, the set $S_{i+1}$ is independent is $\mathcal{M}/(O^{in} \cup \bigcup_{j=1}^{i} X_j)$, so it is also independent in $\mathcal{M}/(O^{in} \cup \bigcup_{j=1}^{i} S_j)$ (as $O^{in} \cup \bigcup_{j=1}^{i} S_j \subseteq O^{in} \cup \bigcup_{j=1}^{i} X_j$) and therefore $O^{in} \cup \bigcup_{j=1}^{i+1} S_j$ is independent in $\mathcal{M}$.

Then, for property (i), consider some $v \in V''$. There exists a unique $i$ so that $v \in X_i$ (as the $X_i$s are disjoint). Hence from the properties of the sampling of $S_i$ we know that $\mathbb{P}[v \in S_i] = 1/\rho$, hence $\mathbb{P}[v \in \tilde{S}] = 1/\rho$. For property (ii), if $v$ and $v'$ are in the same $X_i$, then the way $S_i$ is sampled guarantees that $\mathbb{P}[v \in \tilde{S} \wedge v' \in \tilde{S}] \leq 1/\rho^2$. Otherwise, the choices of $v$ and $v'$ are independent, i.e., $\mathbb{P}[v \in \tilde{S} \wedge v' \in \tilde{S}] = 1/\rho^2$. ◁

For $i = 1, \ldots, r$, let $O_i^{out} = O^{out} \cap (\operatorname{span}_{\mathcal{M}}(\bigcup_{j=1}^{i} X_j) \backslash \operatorname{span}_{\mathcal{M}}(\bigcup_{j=1}^{i-1} X_j))$.

▷ Claim 25. We have $O^{out} = O^{out} \cap \text{span}_{\mathcal{M}}(\bigcup_{i=1}^{r} X_i) = \bigcup_{i=1}^{r} O_i^{out}$.

Proof. The second part of the equality is straightforward, so we focus on the first part. Consider $v \in O \backslash \text{span}_{\mathcal{M}}(\bigcup_{i=1}^{r} X_i)$. It means that when $v$ is processed in Algorithm 1, that element cannot be discarded by the condition in Line 4, because at that time $V' = \bigcup_{j=1}^{i} X_j$ for some $i$ and therefore $v$ is not spanned by $V'$: that element is thereby added to $C$. Hence $v$ is added to $V'$ in the end (Line 11) and $v \in O^{in}$ (more precisely, $v \in O \cap R$, using the notation of equation (1)). ◁

▷ Claim 26. For all $v \in O_i^{out}$, for all $v' \in X_j$ such that $j \leq i$, we have $\deg_w(v) \leq \deg_w(v')$.

Proof. The element $v \in O_i^{out}$ has been discarded after the sets $X_1, \ldots, X_i$ were built (otherwise it would not have been spanned by $V'$, see Line 4), therefore these sets only contain elements having larger or equal weighted degrees. ◁

▷ Claim 27. For all $1 \leq i \leq r$, we have $|\bigcup_{j=1}^{i} O_j^{out}| \leq \sum_{j=1}^{i} k_j$.

Proof. The set $\bigcup_{j=1}^{i} O_j^{out}$ is independent in $\mathcal{M}/O^{in}$ and as it is in $\text{span}_{\mathcal{M}}(\bigcup_{j=1}^{i} X_j)$, it is in $\text{span}_{\mathcal{M}/O^{in}}(\bigcup_{j=1}^{i} X_j)$. Then we obtain $|\bigcup_{j=1}^{i} O_j^{out}| \leq \text{rank}_{\mathcal{M}/O^{in}}(\bigcup_{j=1}^{i} X_j) = \sum_{j=1}^{i} \text{rank}_{\mathcal{M}/(O^{in} \cup \bigcup_{l=1}^{j-1} X_l)}(X_j) = \sum_{j=1}^{i} k_j$, where in the first equality we used Proposition 12 multiple times. ◁

Now we index the elements in $O^{out} = \{o_1, \ldots, o_{|O^{out}|}\}$ so that $O_1^{out} = \{o_1, \ldots, o_{|O_1^{out}|}\}$, $O_2^{out} = \{o_{|O_1^{out}|+1}, \ldots, o_{|O_1^{out}|+|O_2^{out}|}\}$, and so on. Similarly, we index the elements of $\tilde{S} = \{s_1, \ldots, s_{\sum_{i=1}^{r} k_i}\}$ so that $s_1 = s_{1,1}, \ldots, s_{k_1} = s_{1,k_1}$, $s_{k_1+1} = s_{2,1}, \ldots, s_{k_1+k_2} = s_{2,k_2}$, and so on.

In the following, we will say that the element $s_i$ "replaces" the element $o_i$.

▷ Claim 28. For all $1 \leq i \leq |O^{out}|$, we have $\deg_w(o_i) \leq \deg_w(s_i)$.

Proof. Because of Claim 27, an element $o_i \in O_j^{out}$ is replaced by $s_i \in X_{j'}$ for some $j' \leq j$. As a result, by Claim 26, we know that we always have $\deg_w(o_i) \leq \deg_w(v)$ for any $v \in X_{j'}$. As $s_i$ is drawn from $X_{j'}$ we obtain the desired result. ◁

Then, as $S = O^{in} \cup \tilde{S}$, we have:

$$w(E(S)) = w(E(O^{in})) + w(E(\tilde{S})) - w(E(O^{in}, \tilde{S})).$$

We bound $\mathbb{E}[w(E(O^{in}, \tilde{S}))]$ as follows. By construction, $\mathbb{P}[v' \in V''] = 1/\rho$ for all $v' \in V''$. Then we have

$$\begin{aligned}
\mathbb{E}[w(E(O^{in}, \tilde{S}))] &= \sum_{e \in E(O^{in})} w(e) \cdot \mathbb{P}[e \cap \tilde{S} \neq \emptyset] \\
&\leq \sum_{e \in E(O^{in})} \left[ w(e) \cdot \left( \sum_{v' \in e \cap V''} \mathbb{P}[v' \in \tilde{S}] \right) \right] && \text{by union-bound} \\
&= \sum_{e \in E(O^{in})} w(e) \cdot |e \cap V''| \cdot 1/\rho && \text{as } \mathbb{P}[v' \in \tilde{S}] = 1/\rho, \forall v' \in V'' \\
&\leq \sum_{e \in E(O^{in})} w(e) \cdot (\mu - 1) \cdot 1/\rho && \text{as } |e \cap V''| \leq \mu - 1 \\
&= (\mu - 1)/\rho \cdot w(E(O^{in})).
\end{aligned}$$

Furthermore, the value $w(E(\tilde{S}))$ can be rearranged as follows:

$$w(E(\tilde{S})) = \sum_{e\in E(\tilde{S})} \sum_{v\in e\cap\tilde{S}} \frac{w(e)}{|e\cap\tilde{S}|} = \sum_{v\in\tilde{S}} \sum_{e\in\delta(v)} \frac{w(e)}{|e\cap\tilde{S}|}$$
$$= \sum_{v\in\tilde{S}} \sum_{e\in\delta(v)} \left(\left(1 - \frac{|e\cap\tilde{S}|-1}{|e\cap\tilde{S}|}\right)\cdot w(e)\right) = \sum_{v\in\tilde{S}} \left(\deg_w(e) - \sum_{e\in\delta(v)} \frac{|e\cap\tilde{S}|-1}{|e\cap\tilde{S}|}\cdot w(e)\right).$$

Hence $\mathbb{E}[w(E(\tilde{S}))]$ can be written as

$$\mathbb{E}[w(E(\tilde{S}))] = \mathbb{E}\left[\sum_{v\in\tilde{S}} \deg_w(v)\right] - \mathbb{E}\left[\sum_{v\in\tilde{S}} \sum_{e\in\delta(v)} \frac{|e\cap\tilde{S}|-1}{|e\cap\tilde{S}|}\cdot w(e)\right]. \tag{2}$$

We will then focus on upper-bounding the second term, which captures the extent to which edges are counted multiple times in the first term of the sum. We have

$$\begin{aligned}
&\mathbb{E}\left[\sum_{v\in\tilde{S}} \sum_{e\in\delta(v)} \frac{|e\cap\tilde{S}|-1}{|e\cap\tilde{S}|}\cdot w(e)\right] \\
&\leq \mathbb{E}\left[\sum_{v\in V''} \sum_{e\in\delta(v)} w(e)\cdot \mathbb{1}[v\in\tilde{S}\wedge|e\cap\tilde{S}|\geq 2]\right] \\
&= \sum_{v\in V''} \sum_{e\in\delta(v)} w(e)\cdot\mathbb{P}[v\in\tilde{S}\wedge|e\cap\tilde{S}|\geq 2] \\
&\leq \sum_{v\in V''} \sum_{e\in\delta(v)} w(e)\cdot\left(\sum_{v'\in e\cap V''\setminus\{v\}} \mathbb{P}[v\in\tilde{S}\wedge v'\in\tilde{S}]\right) && \text{by union-bound} \\
&\leq \sum_{v\in V''} \sum_{e\in\delta(v)} w(e)\cdot\left(\sum_{v'\in e\cap V''\setminus\{v\}} 1/\rho^2\right) && \text{by Claim 24} \\
&\leq \sum_{v\in V''} \sum_{e\in\delta(v)} w(e)\cdot\left((\mu-1)\cdot 1/\rho^2\right) && \text{as } |e\cap V''\setminus\{v\}|\leq\mu-1 \\
&= (\mu-1)\sum_{v\in V''} 1/\rho^2\cdot\deg_w(v) \\
&= (\mu-1)/\rho\cdot\mathbb{E}\left[\sum_{v\in\tilde{S}}\deg_w(v)\right]. && \text{as } \mathbb{P}[v\in\tilde{S}]=1/\rho,\ \forall v\in V''
\end{aligned}$$

where for the first inequality we apply the worst possible coefficient (namely, $1\geq(\mu-1)/\mu$) each time $e$ is covered more than once by $\tilde{S}$, and for the second inequality we use a union bound, namely, $\mathbb{P}[v\in\tilde{S}\wedge|e\cap\tilde{S}|\geq 2]\leq\sum_{v'\in e\cap V''\setminus\{v\}}\mathbb{P}[v\in\tilde{S}\wedge v'\in\tilde{S}]$. As a result, combining with (2), we obtain

$$\mathbb{E}[w(E(\tilde{S}))]\geq(1-(\mu-1)/\rho)\cdot\mathbb{E}\left[\sum_{v\in\tilde{S}}\deg_w(v)\right].$$

Then this can be compared to $w(E(O^{out}))$ as we have

$$\mathbb{E}\left[\sum_{v\in\tilde{S}}\deg_w(v)\right] \geq \mathbb{E}\left[\sum_{i=1}^{|O^{out}|}\deg_w(s_i)\right] \geq \sum_{i=1}^{|O^{out}|}\deg_w(o_i)\geq w(E(O^{out})),$$

where we use Claim 27 for the first inequality (for $i = r$, i.e., $|\tilde{S}| \geq |O^{out}|$), Claim 28 in the second inequality, and then we use that the sum of the weighted degrees is always greater than the actual weight of covered hyper-edges (because the hype-edges may be counted multiple times in the sum of the weighted degrees) for the last one.

As a result, we finally get:

$$\mathbb{E}[w(E(S))] \geq w(E(O^{in})) + (1-(\mu-1)/\rho)\cdot w(E(O^{out})) - (\mu-1)/\rho \cdot w(E(O^{in})) \\ \geq (1-(\mu-1)/\rho)\cdot w(E(O)).$$

Therefore by averaging principle, there exists $S^* \subseteq V'$ such that $S^* \in \mathcal{I}$ and $w(E(S^*)) \geq (1-(\mu-1)/\rho)\cdot w(E(O))$. ◀

Now we give another interpretation of Algorithm 1 in terms of matroid union, which allows us to give a simpler description of the kernel $V'$. First recall the definition of matroid union:

▶ **Definition 5.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid. Then we can define $\rho\mathcal{M} = (V, \mathcal{I}_\rho)$ as the* union of $\rho$ matroids $\mathcal{M}$, *as follows: $S \in \mathcal{I}_\rho$ if $S$ can be partitioned into $S_1 \cup \cdots \cup S_\rho$ so that for all $i$ we have $S_i \in \mathcal{I}$.*

In Algorithm 2 we provide a simpler description of Algorithm 1 (the equivalence of the two algorithms is proved in Proposition 29).

**Algorithm 2** Algorithm for building a maximum coverage approximate kernel.

```
1: V = {v_1, ..., v_n} where deg_w(v_1) ≥ ··· ≥ deg_w(v_n)
2: V' ← ∅
3: for i = 1, ..., n do  ▷ vertices are processed in non-increasing order of weighted degree
4:     if V' ∪ {v_i} ∈ I_ρ then
5:         V' ← V' ∪ {v_i}
6: return V'
```

▶ **Proposition 29.** *Algorithm 1 and Algorithm 2 build the same kernel $V'$. Moreover, $V'$ is a maximum weight independent set in $\rho\mathcal{M}$ with respect to the weighted degrees.*

**Proof.** To prove the first part of the proposition, it suffices to prove that the condition in Line 4 of Algorithm 1 is equivalent to check whether $V' \cup C \cup \{v_i\}$ is or is not in $\mathcal{I}_\rho$ (if so, then $V' \cup C$ in Algorithm 1 plays the role of $V'$ in Algorithm 2).

Using the decomposition in equation (1), we know that when $v_i$ is processed and we check the condition in Line 4 of Algorithm 1, we have $V' = X_1 \cup \cdots \cup X_j$ for some $j \in \{0, \ldots, r\}$. We know that each $X_l$ is a $\rho$-DBS in $\mathcal{M}/(\bigcup_{l'=1}^{l-1} X_{l'})$, hence each one can be partitioned into $\rho$ independent sets $B_{l,1}, \ldots, B_{l,\rho}$ in $\mathcal{M}/(\bigcup_{l'=1}^{l-1} X_{l'})$, all of size $\mathrm{rank}_{\mathcal{M}/(\bigcup_{l'=1}^{l-1} X_{l'})}(X_l)$. Therefore,

$$V' = \underbrace{(B_{1,1} \cup \cdots \cup B_{j,1})}_{B_1} \cup \cdots \cup \underbrace{(B_{1,\rho} \cup \cdots \cup B_{j,\rho})}_{B_\rho}$$

can be partitioned into $\rho$ independent sets in $\mathcal{M}$, each of size $\mathrm{rank}_{\mathcal{M}}(V')$. Now, regarding condition in Line 4 of Algorithm 1:

- If $v_i \in \text{span}_{\mathcal{M}}(V')$, as $V'$ is a set containing $\rho \cdot \text{rank}_{\mathcal{M}}(X)$ elements, the set $V' \cup \{v_i\}$ will contain $\rho \cdot \text{rank}_{\mathcal{M}}(X) + 1$ elements while still being of rank equal to $\text{rank}_{\mathcal{M}}(V')$: hence it is not possible to partition $V' \cup \{v_i\}$ into $\rho$ independent sets (as each one can contain up to $\text{rank}_{\mathcal{M}}(V')$ elements), so $V' \cup C \cup \{v_i\} \notin \mathcal{I}_\rho$.
- Otherwise, by Claim 22, the densest subset in $C$ with respect to the matroid $\mathcal{M}/V'$ is of density strictly below $\rho$, hence, by Proposition 18, the densest subset in $C \cup \{v_i\}$ is of density at most $\rho$. We can therefore use Theorem 13 to partition $C \cup \{v_i\}$ into $\rho$ independent sets $C_1, \ldots, C_\rho$ in $\mathcal{M}/V'$, hence $V' \cup C \cup \{v_i\} = (B_1 \cup C_1) \cup \cdots \cup (B_\rho \cup C_\rho)$ can be partitioned into $\rho$ independent subsets, i.e., $V' \cup C \cup \{v_i\} \in \mathcal{I}_\rho$.

Therefore Algorithms 1 and 2 build the very same approximate kernel $V'$. As Algorithm 2 is simply the greedy algorithm to build a maximum weight independent set in the matroid $\rho\mathcal{M}$ with respect to the weighted degrees (see [6]), $V'$ is a maximum weight independent set in $\rho\mathcal{M}$. ◀

▶ **Theorem 6.** *Let $\mathcal{M} = (V, \mathcal{I})$ be a matroid and let $f$ be a coverage function on $V$ of frequency bounded by $\mu$. Let $V'$ be a maximum weight independent set in $\rho\mathcal{M}$, with respect to the weights $f(\{v\})$. Then $V'$ contains a $1 - (\mu - 1)/\rho$ approximate solution of the matroid-constrained maximum coverage problem.*

**Proof.** In fact, from the characterization of maximum weight independent sets in [6], any maximum weight independent set in $\rho\mathcal{M}$ can be obtained by choosing the right processing order in Algorithm 2 (i.e., for elements having the same weighted degrees, putting first the ones we want to pick in our kernel). Hence that kernel would have been built following the procedure of Algorithm 1, and therefore the result of Lemma 23 applies. ◀

## References


1 Alexander A. Ageev and Maxim Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *J. Comb. Optim.*, 8(3):307–328, 2004. doi:10.1023/B:JOCO.0000038913.96607.c2.

2 Édouard Bonnet, Vangelis Th. Paschos, and Florian Sikora. Parameterized exact and approximation algorithms for maximum $k$-set cover and related satisfiability problems. *RAIRO Theor. Informatics Appl.*, 50(3):227–240, 2016. doi:10.1051/ita/2016022.

3 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

4 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 575–584. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.60.

5 Jack Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards Sect. B*, 69:67–72, 1965.

6 Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, 1971. URL: http://www.springerlink.com.myaccess.library.utoronto.ca/content/j17526433u12202x/.

7 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

8 Andreas Emil Feldmann, Karthik C. S., Euiwoong Lee, and Pasin Manurangsi. A survey on approximation in parameterized complexity: Hardness and algorithms. *Algorithms*, 13(6):146, 2020. doi:10.3390/a13060146.

**9** Yuval Filmus and Justin Ward. The power of local search: Maximum coverage over a matroid. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 601–612. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.601.

**10** Yuval Filmus and Justin Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM J. Comput.*, 43(2):514–542, 2014. doi:10.1137/130920277.

**11** Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of generalized vertex cover problems. In Frank K. H. A. Dehne, Alejandro López-Ortiz, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures, 9th International Workshop, WADS 2005, Waterloo, Canada, August 15-17, 2005, Proceedings*, volume 3608 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2005. doi:10.1007/11534273_5.

**12** Dorit S Hochbaum and Anu Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics (NRL)*, 45(6):615–627, 1998.

**13** Chien-Chung Huang and François Sellier. Matroid-constrained maximum vertex cover: Approximate kernels and streaming algorithms. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, June 27-29, 2022, Tórshavn, Faroe Islands*, volume 227 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SWAT.2022.27.

**14** Chien-Chung Huang and François Sellier. Matroid-constrained vertex cover. *Theor. Comput. Sci.*, 965:113977, 2023. doi:10.1016/j.tcs.2023.113977.

**15** Chien-Chung Huang and Justin Ward. FPT-algorithms for the $l$-matchoid problem with linear and submodular objectives. *CoRR*, abs/2011.06268, 2020. arXiv:2011.06268.

**16** Naoyuki Kamiyama. A note on robust subsets of transversal matroids. *CoRR*, abs/2210.09534, 2022. doi:10.48550/arXiv.2210.09534.

**17** Daniel Lokshtanov, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. Lossy kernelization. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 224–237. ACM, 2017. doi:10.1145/3055399.3055456.

**18** Pasin Manurangsi. A note on max k-vertex cover: Faster fpt-as, smaller approximate kernel and improved approximation. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 15:1–15:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.15.

**19** Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum $k$-coverage, unique set cover and related problems (via $t$-wise agreement testing theorem). In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 62–81. SIAM, 2020. doi:10.1137/1.9781611975994.5.

**20** Dániel Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008. doi:10.1093/comjnl/bxm048.

**21** Andrew McGregor, David Tench, and Hoa T. Vu. Maximum coverage in the data stream model: Parameterized and generalized. In Ke Yi and Zhewei Wei, editors, *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*, volume 186 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICDT.2021.12.

**22** George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

**23** Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

**24** Piotr Skowron. FPT approximation schemes for maximizing submodular functions. *Inf. Comput.*, 257:65–78, 2017. doi:10.1016/j.ic.2017.10.002.

**25** Piotr Krzysztof Skowron and Piotr Faliszewski. Fully proportional representation with approval ballots: Approximating the maxcover problem with bounded frequencies in FPT time. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2124–2130. AAAI Press, 2015. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9674`.

**26** Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 588–597. IEEE Computer Society, 2001. `doi:10.1109/SFCS.2001.959935`.

# Fault Tolerance in Euclidean Committee Selection

**Chinmay Sonar**
Department of Computer Science, University of California, Santa Barbara, CA, USA

**Subhash Suri**
Department of Computer Science, University of California, Santa Barbara, CA, USA

**Jie Xue**
Department of Computer Science, New York University, Shanghai, China

## Abstract

In the committee selection problem, the goal is to choose a subset of size $k$ from a set of candidates $C$ that collectively gives the best representation to a set of voters. We consider this problem in Euclidean $d$-space where each voter/candidate is a point and voters' preferences are implicitly represented by Euclidean distances to candidates. We explore *fault-tolerance* in committee selection and study the following three variants: (1) given a committee and a set of $f$ failing candidates, find their optimal replacement; (2) compute the worst-case replacement score for a given committee under failure of $f$ candidates; and (3) design a committee with the best replacement score under worst-case failures. The score of a committee is determined using the well-known (min-max) Chamberlin-Courant rule: minimize the maximum distance between any voter and its closest candidate in the committee. Our main results include the following: (1) in one dimension, all three problems can be solved in polynomial time; (2) in dimension $d \geq 2$, all three problems are NP-hard; and (3) all three problems admit a constant-factor approximation in any fixed dimension, and the optimal committee problem has an FPT bicriterion approximation.

**2012 ACM Subject Classification** Theory of computation → Randomness, geometry and discrete structures

**Keywords and phrases** Multiwinner elections, Fault tolerance, Geometric Hitting Set, EPTAS

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.95

**Related Version** *Full Version*: `https://arxiv.org/abs/2308.07268`

## 1 Introduction and Problem Statement

We consider the computational complexity of adding fault tolerance into spatial voting. In spatial voting [7, 1, 29], the voters and the candidates are both modeled as points in some $d$-dimensional space, where each dimension represents an independent *policy issue* that is important for the election, and each voter's preference among the candidates is implicitly encoded by a distance function. For example, in the simplest 1-dimensional setting, voters and candidates are points on a line indicating their real-valued preference on a single issue. The specific setting for our work is *multiwinner* spatial elections, also called *committee selection*, in $d$ dimensions where we have a set $V$ of $n$ voters, a set $C$ of $m$ candidates, and a committee size (integer) $k$. The goal is to choose a subset of $k$ candidates, called the *winning committee*, that collectively best represents the preferences of all the voters [9, 11, 12].

One aspect of committee selection that appears not to have been investigated is *fault tolerance*, that is, how robust a chosen committee is against the possibility that some of the winning members may default. Committee selection problems model a number of applications in the social sciences and in computer science where such defaults are not uncommon, such as democratic elections, staff hiring, choosing public projects, locations of public facilities, jury selection, cache management, etc. [21, 14, 26, 2, 4, 23, 13]. In this paper, we are particularly interested in designing algorithms to address questions of the following kind: If some of the winning members default, how badly does this affect the overall score of the committee? Or,


how much does the committee score suffer if a *worst-case* subset of size $f$ defaults? Finally, can we *proactively* choose a committee in such a way that it can tolerate up to $f$ faults with the minimum possible score degradation? We begin by formalizing these problems more precisely and then describing our results.

Suppose $V = \{v_1, \ldots, v_n\}$ is a set of $n$ voters and $C = \{c_1, \ldots, c_m\}$ is a set of $m$ candidates, modeled as points in $d$-dimensional Euclidean space. (We occasionally call the tuple $(V, C)$ an election $E$.) Given a positive integer $k$, we want to elect $k$ candidates, called the *committee*, using the well-known Chamberlin-Courant voting rule [5]. This rule assigns a score to each committee as follows. Let $T \subseteq C$ be a committee. For each voter $v$, the score of $T$ for $v$ is defined as $\sigma(v, T) = \min_{c \in T} d(v, c)$, namely, the distance from $v$ to its closest candidate in $T$.[1] The *score* of the committee $T$ is defined as $\sigma(T) = \max_{v \in V} \sigma(v, T)$, namely, the largest distance between any voter and its closest neighbor in $T$. (In facility location parlance, this is the well-known $k$-center problem.)

The fault tolerance of a committee is parameterized by a positive integer $f$, which is the upper bound on the number of candidates that can fail.[2] Throughout the paper, we use the notation $J$ to denote a failing set of candidates. We are allowed to replace the failing members of $J$ with any set of at most $|T \cap J|$ candidates from $C \setminus J$. We often denote this set of *replacement* candidates by $R$. However, *we must keep all the non-failing members of $T$ in the committee* – that is, the replacement committee is the set $(T \setminus J) \cup R$ – and throughout the paper our goal is to optimize this committee's score, namely $\sigma((T \setminus J) \cup R)$.

We consider the following three versions of fault-tolerant committee selection, presented in increasing order of complexity. The first problem is the simplest: given a committee and a failing set, find the best replacement committee.

**Optimal Replacement Problem** (ORP)
**Input:** An election $E = (V, C)$, a committee $T \subseteq C$ and a failing set $J \subseteq C$.
**Goal:** Find a replacement set $R \subseteq C \setminus J$ of size at most $|T \cap J|$ minimizing $\sigma((T \setminus J) \cup R)$.

Our second problem is to quantify the fault tolerance of a given committee $T$ over worst-case faults. That is, what is the largest score of $T$'s replacement when a worst-case subset of $f$ faults occur? We introduce the following notation as $T$'s measure of $j$-fault-tolerance, for any $0 \leq j \leq f$: $\sigma_j(T) = \max_{J \subseteq C \text{ s.t. } |J| \leq j} \sigma((T \setminus J) \cup R)$, where $R$ is an optimal replacement set with size at most $|T \cap J|$. We want to compute $\sigma_f(T)$. Occasionally, we also use the notation $\sigma_0(T)$ for the no-fault score of $T$, namely $\sigma(T)$.

**Fault-Tolerance Score** (FTS)
**Input:** An election $E = (V, C)$, a committee $T \subseteq C$ and a fault-tolerance parameter $f$.
**Goal:** Compute $\sigma_f(T)$.

Our third and final problem is to compute a committee with optimal fault-tolerance score.

**Optimal Fault-Tolerant Committee** (OFTC)
**Input:** An election $E = (V, C)$, a committee size $k$ and a fault-tolerance parameter $f$.
**Goal:** Find $T \subseteq C$ of size at most $k$ minimizing $\sigma_f(T)$.

[1] Originally, Chamberlin and Courant [5] defined a voting rule on Borda scores (also known as Borda-CC). In this paper, similarly to [2], we study a min-max version of this rule on a more general scoring function, which in our case is based on voter-candidate distances.

[2] In our work, we will allow any subset of size $f$ from $C$ to fail, so the faults can also include candidates not in the selected committee $T$. This only makes the problem harder because the adversary can always limit the faults to $T$, and elimination of candidates from $C \setminus T$ makes finding replacements for failing committee members more difficult.

## 1.1 Our Results

We first show that even in one dimension, fault-tolerant committee problems are nontrivial. In particular, while the Optimal Replacement Problem (ORP) is easily solved by a simple greedy algorithm, the other two problems, Fault-Tolerance Score (FTS) and Optimal Fault-Tolerant Committee (OFTC), do not appear to be easy. Our main result in one dimension is the design of efficient dynamic-programming-based algorithms for these two problems. Along the way, we solve a *fault-tolerant* Hitting Set problem for points and unit intervals, which may be of independent interest.

In two dimensions and higher, OFTC is NP-hard because of its close connection to the $k$-center problem. However, we show that even the seemingly simpler problem of optimal replacement (ORP) is also NP-hard. Our main results include a constant-factor approximation for all three problems in any fixed dimension (in fact, in any metric space), as well as a novel bicriterion FPT approximation via an EPTAS whose running time has the form $f(\epsilon)n^{\mathcal{O}(1)}$. For ease of reference, we show these results in the following table.

**Table 1** Summary of our results.

| | **One-dimensional instances** | **Dimension $d \geq 2$** | | |
|---|---|---|---|---|
| | | **Complexity** | **Approximation** | **Bounded $f$** |
| **ORP** | P (Theorem 1) | NP-hard (Theorem 11) | 3-approx. (Lemma 12) | P (full version) |
| **FTS** | P (Theorem 6) | NP-hard (Theorem 11) | 3-approx. (Lemma 13) | P (full version) |
| **OFTC** | P (Theorem 9) | NP-hard (Theorem 11) | 5-approx. (Lemma 17)<br>Bicriterion-EPTAS (Theorem 20) | NP-hard (full version)<br>3-approx. (Theorem 16) |

Due to limited space, proofs of some of the theorems/lemmas (marked with ($\star$)) are deferred to the full version of the paper.

## 1.2 Related Work

To the best of our knowledge, the issue of fault tolerance in committee selection has not been studied in voting literature – their primary focus is on protocols and algorithms for choosing candidates [11, 12, 25, 27, 28, 8, 3]. However, the following two lines of work consider some related issues. First, in the "unavailable candidate model" [22, 15] the goal is to choose a *single winner* with maximum expected score when candidates fail according to a given probability distribution; in contrast, we consider multiwinner elections under worst-case faults. In the second line of work, a set of election control problems are considered where candidates are added [10] or deleted [17] to change the outcome of the election. In this setting, the candidate set is modified to obtain a favorable election outcome, which is a rather different problem than ours.

In the facility-location research, there has been prior work on adding fault tolerance to $k$-center or $k$-median solutions [6, 20, 19, 30, 16], but the main approach there is to assign each user (voter) to multiple facilities (candidates). In particular, the "$p$-neighbor $k$-center" framework [6] minimizes the maximum distance between a user and its $p$th center as a way to protect against $p-1$ faults. This formulation, however, differs from our optimal fault-tolerant committee problem (OFTC) because in our setting the replacement candidates are chosen *after* failing candidates are announced. Therefore, in the OFTC problem, the

designer does not have to *simultaneously* allocate $p$ neighbors for all the voters. Furthermore, to the best of our knowledge, neither of our first two problems – Optimal Replacement (ORP) and Fault-Tolerance Score (FTS) – have been studied in the facility-location literature, and initiate a new research direction. We also formulate and solve a fault-tolerant hitting set problem in one dimension, which may be of independent interest.

## 2 Fault-Tolerant Committees in One Dimension

Even in one dimension, computing the fault-tolerance score of a given committee or finding a committee with minimum fault-tolerance score is nontrivial. The optimal replacement problem, however, is easy – a simple greedy algorithm works. Our main result in this section is to design efficient dynamic-programming algorithms for the former two problems. In doing so, we also solve the fault-tolerant version of Hitting Set for points and unit segments.

### 2.1 Optimal Replacement Problem

In the Optimal Replacement Problem (ORP), we are given a committee $T \subseteq C$ and a failing set $J \subseteq C$, and we must find a replacement set $R$ minimizing the score $\sigma((T \setminus J) \cup R)$, where $|R| \leq |T \cap J|$. Since this score is always the distance between some voter-candidate pair, it suffices to solve the following decision problem: Is there a replacement set with score at most $r$? We can then try all possible $O(nm)$ distances to find the smallest feasible replacement score.

This decision problem is equivalent to the following hitting set problem: for each voter $v \in V$, let $I_v$ be the interval of length $2r$ centered at $v$, and let $\mathcal{I} = \{I_v : v \in V\}$ be the set of these $n$ (voter) intervals. A subset of candidates is a hitting set for $\mathcal{I}$ if each interval contains at least one of the candidates. In our problem, we are given a hitting set $T$ and a failing subset of candidates $J$, and we must find the minimum-size replacement hitting set. Such a replacement is easily found using the standard greedy algorithm, as follows. We first remove all of the intervals from $\mathcal{I}$ that are already hit by a candidate in $T \setminus J$, and we also remove all the failing candidates $J$ from $C$. For the leftmost remaining interval, we then choose the rightmost candidate $c$ contained in it, add it to $R$, delete all intervals hit by $c$, and iterate until all remaining intervals are hit. If we ever encounter an interval containing no candidate, or if the size of the replacement set is larger than $|T \setminus J|$, the answer to the decision problem is no. Otherwise, the solution is $R$. The greedy algorithm is easily implemented to run in time $\mathcal{O}((m+n)\log(m+n))$. To find the optimal replacement set, we can do a binary search over $O(nm)$ values of $r$ and find the smallest $r$ for which $|(T \setminus J) \cup R| \leq k$.

▶ **Theorem 1.** *The Optimal Replacement Problem can be solved in time $\mathcal{O}((m+n)\log^2(m+n))$ for one-dimensional Euclidean elections.*

### 2.2 Computing the Fault-Tolerance Score (FTS) of a Committee

We now come to the more difficult problem of computing the fault-tolerance score $\sigma_f(T)$ of a committee $T$ in one dimension, which is the worst case over all possible failing sets of $T$. Once again it suffices to solve the following decision problem: given a size-$k$ committee $T$ and a real number $r$, can we find a replacement with score at most $r$ for *every* failing subset of size $f$? Using our hitting set formulation, $\sigma_f(T) \leq r$ if and only if $T$ is an *$f$-tolerant hitting set* of $\mathcal{I}$, that is, for any failing set $J \subseteq C$ of size at most $f$, there exists a replacement set $R \subseteq C \setminus J$ such that $|(T \setminus J) \cup R| \leq |T|$ and $(T \setminus J) \cup R$ hits $\mathcal{I}$. (Recall that each member of $\mathcal{I}$ is an interval of length $2r$ centered at one of the voter positions.) We can then compute the fault-tolerance score of $T$ by trying each of the $O(nm)$ voter-candidate distances to find the smallest $r$ for which this decision problem has a positive answer.

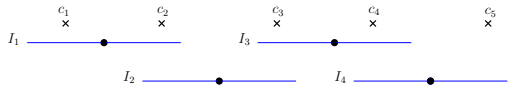



**Figure 1** The figure shows an interval hitting set instance with four intervals and five points. The set $\{c_2, c_4\}$ is a feasible hitting set. For $X = \{c_2, c_3, c_5\}$, the intervals $I_1, I_3, I_4$ are $X$-disjoint.

We solve this fault-tolerant hitting set decision problem by observing that the size of a *smallest* hitting set equals the size of a *maximum* independent set, defined with respect to candidate points and voter intervals in the following way. Suppose the intervals of $\mathcal{I} = \{I_1, \ldots, I_n\}$ are sorted left to right. First, we can assume without loss of generality that $|I_i \cap C| > f$ for all $i \in [n]$, since otherwise there is no $f$-tolerant hitting set for $\mathcal{I}$. Given a set of points $X$ in $\mathbb{R}$, we say that a set of intervals is *$X$-disjoint* if each point in $X$ is contained in at most one interval. (That is, $X$-disjoint intervals can be thought of as *independent* in that they contain disjoint sets of points in $X$). The following claim is easy to prove.

▶ **Lemma 2.** *Given a set of points $X$ and a set of intervals $\mathcal{J}$ on the real line, the size of a minimum hitting set $X' \subseteq X$ of $\mathcal{J}$ equals the maximum size of an $X$-disjoint subset of $\mathcal{J}$.*

Thus, if $T \subseteq C$ is an $f$-tolerant hitting set for $\mathcal{I}$, then for any failing set $J \subseteq C$, the size of any $(C \setminus J)$-disjoint subset of $\mathcal{I}$ is at most $|T|$. One should note that the size of the maximum $(C \setminus J)$-disjoint subset in $\mathcal{I}$ is a monotonically increasing function of $|J|$ – as more candidates fail, more intervals can become disjoint. Our goal is to find the maximum size of such a disjoint interval family over all possible failure sets $J$ of size at most $f$. We will do this using dynamic programming, by combining solutions of subproblems, where each subproblem corresponds to an index range $[i, j]$, over the set of candidate points $c_1, \ldots, c_m$. Assuming that the candidate points $C = \{c_1, \ldots, c_m\}$ are ordered from left to right, our subproblems are defined as follows, for $1 \leq i \leq j \leq m$:

- $C_{i,j} = \{c_i, \ldots, c_j\}$ is the set of candidates in the range $[c_i, c_j]$.
- $\mathcal{I}_{i,j} = \{I \in \mathcal{I} : I \cap C \subseteq C_{i,j}\}$ is the set of intervals that only contain points from $C_{i,j}$.
- For any $J \subseteq C_{i,j}$, $\delta_{i,j}(J)$ is the maximum size of a $(C_{i,j} \setminus J)$-disjoint subset of $\mathcal{I}_{i,j}$.
- The subproblems we want to solve are the values $\delta_{i,j}(f) = \max_{J \subseteq C_{i,j}, |J| \leq f} \delta_{i,j}(J)$.

The key technical lemma of this section is the following claim.

▶ **Lemma 3.** *$T \subseteq C$ is an $f$-tolerant hitting set of $\mathcal{I}$ if and only if $|T \cap C_{i,j}| \geq \delta_{i,j}(f)$, for all $1 \leq i \leq j \leq m$.*

**Proof.** We first show the "if" part of the lemma. Assume $|T \cap C_{i,j}| \geq \delta_{i,j}(f)$ for all $i, j \in [m]$ with $i \leq j$. To see that $T$ is an $f$-tolerant hitting set of $\mathcal{I}$, consider a failing set $J \subseteq C$ of size at most $f$. We have to show the existence of a replacement set $R \subseteq C \setminus J$ such that $|(T \setminus J) \cup R| \leq |T|$ and $(T \setminus J) \cup R$ is a hitting set of $\mathcal{I}$. We write $T \setminus J = \{c_{i_1}, \ldots, c_{i_p}\}$, where $i_1 < \cdots < i_p$. For convenience, set $i_0 = 0$ and $i_{p+1} = m + 1$. By our assumption, every interval $I \in \mathcal{I}$ is hit by some point in $C$. Thus, either $I$ is hit by $T \setminus J$ or $I$ belongs to $\mathcal{I}_{i,j}$ where $i = i_{t-1} + 1$ and $j = i_t - 1$ for some index $t \in [p + 1]$. Now consider an index $t \in [p + 1]$. We write $T_t = T \cap C_{i,j}$ and define $R_t \subseteq C_{i,j} \setminus J$ as a minimum hitting set of $\mathcal{I}_{i,j}$. By Lemma 2, the size of $R_t$ is equal to the maximum size of a $(C_{i,j} \setminus J)$-disjoint subset of $\mathcal{I}_{i,j}$, which is nothing but $\delta_{i,j}(J \cap C_{i,j})$. Also, by assumption, we have $|T_t| = |T \cap C_{i,j}| \geq \delta_{i,j}(f) \geq \delta_{i,j}(J \cap C_{i,j})$. Therefore, $|R_t| \leq |T_t|$. Finally, we define $R = \bigcup_{t=1}^{p+1} R_t$. Clearly, $(T \setminus J) \cup R$ hits $\mathcal{I}$. So it suffices to show that $|(T \setminus J) \cup R| \leq |T|$. Since $|R_t| \leq |T_t|$ for all $t \in [p + 1]$, we have

$$|(T\backslash J) \cup R| = |T\backslash J| + \sum_{t=1}^{p+1} |R_t| \leq |T\backslash J| + \sum_{t=1}^{p+1} |T_t| = |T|,$$

which completes the proof of the "if" part.

Next, we prove the "only if" part of the lemma. Assume $T \subseteq C$ is an $f$-tolerant hitting set of $\mathcal{I}$. Consider two indices $i, j \in [m]$ with $i \leq j$. To show $|T \cap C_{i,j}| \geq \delta_{i,j}(f)$, it suffices to show that $|T \cap C_{i,j}| \geq \delta_{i,j}(J)$ for all $J \subseteq C_{i,j}$ with $|J| \leq f$. Since $T$ is an $f$-tolerant hitting set of $\mathcal{I}$, there exists $R \subseteq C\backslash J$ such that $|(T\backslash J) \cup R| \leq |T|$ and $(T\backslash J) \cup R$ is a hitting set of $\mathcal{I}$. For brevity, let $T' = (T\backslash J) \cup R$. By definition, the intervals in $\mathcal{I}_{i,j}$ can only be hit by the points in $C_{i,j}$. Thus, $T' \cap C_{i,j}$ is a hitting set of $\mathcal{I}_{i,j}$. As $T' \cap C_{i,j} \subseteq C_{i,j}\backslash J$, by Lemma 2, the size of $T' \cap C_{i,j}$ is at least the maximum size of a $(C_{i,j}\backslash J)$-disjoint subset of $\mathcal{I}_{i,j}$, i.e., $|T' \cap C_{i,j}| \geq \delta_{i,j}(J)$. Furthermore, because $J \subseteq C_{i,j}$, we have $(T\backslash J)\backslash C_{i,j} = T\backslash C_{i,j}$. It follows that $T\backslash C_{i,j} \subseteq T'\backslash C_{i,j}$ and thus $|T\backslash C_{i,j}| \leq |T'\backslash C_{i,j}|$. For a committee $T$, we can partition $T$ into two parts: the part containing candidates in $C_{i,j}$ and the part containing candidates outside of $C_{i,j}$. Hence, $|T| = |T \cap C_{i,j}| + |T\backslash C_{i,j}|$ and $|T'| = |T' \cap C_{i,j}| + |T'\backslash C_{i,j}|$. Because $|T'| \leq |T|$ and $|T'\backslash C_{i,j}| \geq |T\backslash C_{i,j}|$, we have $|T' \cap C_{i,j}| \leq |T \cap C_{i,j}|$. Therefore, $|T \cap C_{i,j}| \geq \delta_{i,j}(J)$. This completes the proof of Lemma 3. ◀

In order to decide if $\sigma_f(T) \leq r$, therefore, we just have to compute $\delta_{i,j}(f)$, for all $i, j$, and check the condition $|T \cap C_{i,j}| \geq \delta_{i,j}(f)$. We now show how to do that efficiently.

### Efficiently Computing $\delta_{i,j}(f)$

For ease of presentation, we show how to compute $\delta_{1,m}(f)$; computing other $\delta_{i,j}(f)$ is similar. We have $C_{1,m} = C$, $\mathcal{I}_{1,m} = \mathcal{I}$, and $\delta_{1,m}(f)$ is size of the largest subset of $\mathcal{I}$ that is $(C\backslash J)$-disjoint for any failing set $J \subseteq C$ with $|J| \leq f$. The intervals of $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ are in the left to right sorted order and, for each $i \in [n]$, let $C(I_i) = C \cap I_i$ be the set of points in $C$ that hits $I_i$. Define $\Gamma[i][j]$ as the maximum size of an $(C \setminus X)$-disjoint subset $\mathcal{J} \subseteq \{I_1, \ldots, I_i\}$ such that $X \subseteq C$ and $|X| \leq j$.

▶ **Lemma 4.** *We have the following recurrence*

$$\Gamma[i][j] \;=\; \max \left\{ \begin{array}{l} \Gamma[i-1][j], \\ \max\limits_{0 \leq i' \leq i} 1 + \Gamma[i'][j - |C(I_i) \cap C(I_{i'})|] \end{array} \right\}$$

Clearly, $\delta_{1,m}(f) = \Gamma[n][f]$. The base case for our dynamic program is $\Gamma[0, j] = 0$ for all $j \in [f]$ and $\Gamma[i][j] = -\infty$ for $j < 0$ and all $i \in [n]$. Our dynamic program runs in time $\mathcal{O}(n^2 m f)$. In the same way, we can compute the values of $\delta_{i,j}(f)$ for all $i, j \in [m]$ with $i \leq j$.

▶ **Lemma 5.** *$\delta_{i,j}(f)$, for all $1 \leq i \leq j \leq m$, can be computed in time $\mathcal{O}(n^2 m^3 f)$.*

Given a hitting set $T \subseteq C$ and the values $\delta_{i,j}(f)$, we can verify the condition in Lemma 3 in time $\mathcal{O}(m^3)$. We can then use binary search to find the smallest value of $r$ for which $T$ is an $f$-tolerant hitting set. This establishes the following result.

▶ **Theorem 6.** *The fault-tolerance score of a 1-dimensional committee $T$ can be computed in time $\mathcal{O}(n^2 m^3 f \log(nm))$.*

## 2.3 Optimal Fault-Tolerant Committee

We now address the problem of *designing* a fault-tolerant committee: select a committee $T$ of size $k$ whose fault-tolerance score $\sigma_f(T)$ is minimized. Thus, our goal is *not* to optimize the fault-free score of $T$, namely $\sigma_0(T)$, but rather the score that the best replacement will have after a worst-case set of $f$ faults in $T$, namely $\sigma_f(T)$. Following the earlier approach, we again focus on the decision question: given some $r \geq 0$, is there a committee of size $k$ with $\sigma_f(T) \leq r$? For a given value of $r$, we construct our hitting set instance with candidate-points and voter-intervals, and compute a minimum-sized $f$-tolerant hitting set $T \subseteq C$ as follows:

1. Compute the value of $\delta_{i,j}(f)$, for all $1 \leq i \leq j \leq m$.
2. Compute a minimum subset $T \subseteq C$ satisfying $|T \cap C_{i,j}| \geq \delta_{i,j}(f)$, for all $1 \leq i \leq j \leq m$.
3. If $|T| \leq k$, we have a solution; otherwise, the answer to the decision problem is no.

Step (1) is implemented using the dynamic program of the previous subsection, and so it suffices to explain how to implement step (2). We assume without loss of generality that $|C_{i,j}| \geq \delta_{i,j}(f)$ for all $i, j$, because otherwise there is no solution. We compute a set $T$ using the following greedy algorithm.

- Initialize $T = \emptyset$.
- For each $c_k$ for $k \in [m]$, if there exists $i, j \in [m]$ with $i \leq k \leq j \leq m$ such that $\delta_{i,j}(f) \geq |T \cap C_{i,j}| + (j - k + 1)$, then add $c_k$ to $T$.

The algorithm runs in time $\mathcal{O}(m^3)$. To prove correctness, we first claim the following.

▶ **Lemma 7.** *$|T \cap C_{i,j}| \geq \delta_{i,j}(f)$, for all $1 \leq i \leq j \leq m$.*

**Proof.** Suppose not, so we have $|T \cap C_{i,j}| < \delta_{i,j}(f)$, for some $i \leq j$. We recall that for any interval $I_i \in \mathcal{I}$, $|I_i \cap C| > f$. Therefore, for any failing set $J$, $C_{i,j} \setminus J$ is a hitting set of $\mathcal{I}_{i,j}$, and $|C_{i,j}| \geq \delta_{i,j}(f)$. This implies that there exists some point among $c_i, \ldots, c_j$ that is not in $T$. Let $k \in \{i, \ldots, j\}$ be the largest index such that $c_k \notin T$. For convenience, we use $T'$ to denote the set $T$ in the iteration of our algorithm that considers $c_k$. Note that $T \cap C_{i,j} = (T' \cap C_{i,j}) \cup \{c_{k+1}, \ldots, c_j\}$ and $(T' \cap C_{i,j}) \cap \{c_{k+1}, \ldots, c_j\} = \emptyset$. Therefore, $|T' \cap C_{i,j}| = |T \cap C_{i,j}| - (j-k) < \delta_{i,j}(f) - (j-k)$. This implies $|T' \cap C_{i,j}| + (j-k+1) \leq \delta_{i,j}(f)$. By our algorithm, in this case we should include $c_k$ in $T$, which contradicts the fact that $c_k \notin T$. ◀

We now argue that $T$ has the minimum size among all subsets of $C$ satisfying the property of Lemma 7. Let $\mathsf{opt}$ be the minimum size of a subset of $C$ satisfying the desired property. We write $T = \{c_{k_1}, \ldots, c_{k_r}\}$, where $k_1 < \cdots < k_r$.

▶ **Lemma 8 (⋆).** *For any $t \in [r]$, there exists a subset $T^* \subseteq C$ such that (1) $|T^* \cap C_{i,j}| \geq \delta_{i,j}(f)$ for all $i, j \in [m]$ with $i \leq j$, (2) $|T^*| = \mathsf{opt}$, and (3) $\{c_{k_1}, \ldots, c_{k_t}\} \subseteq T^*$.*

We use binary search to find the smallest $r$ such that the reduced instance has an $f$-tolerant hitting set of size at most $k$. Therefore, the following theorem holds.

▶ **Theorem 9.** *Optimal Fault-Tolerant Committee can be solved in time $\mathcal{O}(n^2 m^3 f \log(nm))$ for one-dimensional Euclidean elections.*

▶ Remark 10. Our dynamic programming algorithm works as long as either the set $V$ or the set $C$ is embedded in $\mathbb{R}$ (i.e., has a linear ordering), while the other set can have an arbitrary $d$-dimensional embedding. Moreover, we can also extend our algorithms to ordinal elections with (widely studied) single-peaked preferences [2, 24] to compute an optimal fault-tolerant Chamberlin-Courant committee.

# 3 Fault-Tolerant Committees in Multidimensional Space

We now consider fault tolerance in multidimensional elections. Unsurprisingly, the optimal committee design problem is intractable – it is similar to facility location – but it turns out that the seemingly simpler variants ORP and FTS are also intractable. In particular, we have the following (refer to the full version of the paper for details).

▶ **Theorem 11** (⋆). *All three problems (Optimal Replacement, Fault-Tolerance Score, and Optimal Fault-Tolerant Committee) are NP-hard, in any dimension $d \geq 2$ under the Euclidean norm, where size of the committee $k$ and the failure parameter $f$ are part of the input.*

## 3.1 Optimal Replacement Problem

A simple greedy algorithm achieves a 3-approximation for the Optimal Replacement Problem in any fixed dimension $d$ as well as in any metric space.

▶ **Lemma 12.** *We can find a 3-approximation for ORP in time $O(k(nk+m))$.*

**Proof.** Let $T \subseteq C$ be the given committee and let $J \subseteq T$ be the failing set. In order to find the replacement set $R$, we initialize $\hat{T} = T \setminus J$, and then repeat the following two steps $|T \cap J|$ times: (1) Choose the farthest voter from $\hat{T}$, namely, choose $\hat{v} = \arg\max_{v \in V} d(v, \hat{T})$, and (2) Add to $\hat{T}$ the candidate $\hat{c} \notin \hat{T}$ that is closest to $\hat{v}$. Upon termination, we clearly have $|\hat{T}| = |T|$. Due to limited space, the proof of the approximation ratio is deferred to the full version of our paper. ◀

## 3.2 Computing the Fault-Tolerance Score

We can also approximate the optimal fault-tolerance score of a committee within a factor of 3. Specifically, if the optimal fault-tolerance score of $T$ is $\sigma_f(T) = \sigma^*$, then our algorithm returns a real number $\sigma'$ such that $\sigma^* \leq \sigma' \leq 3\sigma^*$.

For each voter $v$, let $d_f(v)$ be $v$'s distance to its $(f+1)^{th}$ closest candidate, and let $d' = \max_{v \in V} d_f(v)$ be the maximum of these values over all voters. The basic idea behind our approximation is simple and uses the following two facts: (1) $\sigma^* \geq d'$, and (2) $\sigma^* \geq \sigma(T)$. The first one holds because $d'$ is the best score possible if some voter's $f$ nearest candidates fail, and the second one holds because a failure can only worsen the score (that is, $\sigma_f(T) \geq \sigma(T)$ for any $f > 0$). Therefore, the distance $\sigma' = d' + 2\sigma(T)$ is clearly within a factor of 3 of the optimal $\sigma^*$. We claim that for any failing set $J \subseteq C$, there exists a replacement $R \subseteq C \setminus J$ of size at most $|T \cap J|$ such that $\sigma((T \setminus J) \cup R) \leq \sigma'$. Due to limited space, we omit the proof from the extended abstract; it is included in the full version of our paper.

▶ **Lemma 13.** *The fault-tolerance score of a committee can be approximated within a factor of 3 in time $\mathcal{O}(nm \log(f))$.*

## 3.3 Optimal Fault-Tolerant Committee

We now discuss how to design approximately optimal fault-tolerant committees in multiwinner elections. Specifically, given a set of voters $V$ and a set of candidates $C$ in $d$-space, along with parameters $k$ (committee size) and $f$ (number of faults), we want to compute a size $k$ committee $T \subseteq C$ with the minimum fault-tolerance score $\sigma_f(T)$. We prove two approximation results for this problem: (1) We can solve this problem within an approximation factor of 3 in polynomial time if the parameter $f$ is treated as a constant (while $k$ remains possibly

unbounded). If $f$ is not assumed to be a constant, we can solve the problem within an approximation factor of 5. (2) We give an EPTAS with running time $(1/\varepsilon)^{O(1/\varepsilon^{2d})}(m+n)^{O(1)}$ which is a *bicriterion* approximation, where the output committee $T$ is fault-tolerant for at least $(1-\varepsilon)n$ voters with $\sigma_f(T) \leq (1+\varepsilon)\sigma^*$. The next two subsections discuss these results.

### 3.3.1 3-Approximation for Bounded $f$

Let $\sigma^*$ be the optimal $f$-tolerant score of a committee of size $k$. We compute the approximation solution via an approximate decision algorithm, which takes as input a number $\sigma \geq \sigma_f(C)$ and returns a committee $T \subseteq C$ of size at most $k$ with $\sigma_f(T) \leq 3\sigma$ if $\sigma \geq \sigma^*$. (We slightly abuse notation to introduce a convenient quantity $\sigma_f(C)$, which is the $f$-fault-tolerance score of a committee with all the input candidates. This is clearly a lower bound on any size $k$ committee's score.)

For a committee $T \subseteq C$ and a failing set $J \subseteq C$, let $\delta(T, J)$ denote the score obtained after finding an optimal replacement $K$. That is,

$$\delta(T, J) = \min_{K \in C \setminus J, |K| = |T \cap J|} \sigma_0((T \setminus J) \cup K).$$

Thus, $\sigma_f(T) = \max_{J \subseteq C, |J| \leq f} \delta(T, J)$. Our approximation algorithm is shown in Algorithm 1. It begins with an empty committee $T$ (line 1), and as long as there exists a failing set $J$ of size at most $f$ for which $\delta(T, J) > 3\sigma$, [3] we do the following.

First, we remove all candidates in $J$ from $T$ (line 3). Then, whenever there exists a voter $v \in V$ with $d(v, T) > 3\sigma$, we add to $T$ a candidate $c \in C \setminus J$ whose distance to $v$ is at most $\sigma$ (lines 5-6). Such a $c$ always exists because $\sigma$ is at least the distance to the $(f+1)^{th}$ closest neighbor to $v$.

We call this voter $v$ the *witness* of $c$, denoted by $\mathsf{wit}[c]$ (line 7). Adding $c$ to $T$ guarantees that $d(v, T) \leq \sigma$. We repeat this procedure (the inner while loop) until $d(v, T) \leq 3\sigma$ for all $v \in V$. Finally, the outer while loop terminates when $\delta(T, J) \leq 3\sigma$ for all $J \subseteq C$ of size at most $f$, i.e., $\sigma_f(T) \leq 3\sigma$. At this point, we return the committee $T$.

**Algorithm 1** Approximate decision algorithm.

**Input:** a set $V$ of voters, a set $C$ of candidates, the committee size $k$, the fault-tolerance parameter $f$, and a number $\sigma \geq \sigma_f(C)$

1: $T \leftarrow \emptyset$
2: **while** $\exists\ J \subseteq C$ such that $|J| \leq f$ and $\delta(T, J) > 3\sigma$ **do**
3: $\quad T \leftarrow T \setminus J$
4: $\quad$ **while** $\exists\ v \in V$ such that $d(v, T) > 3\sigma$ **do**
5: $\quad\quad c \leftarrow$ a candidate in $C \setminus J$ satisfying $d(v, c) \leq \sigma$
6: $\quad\quad T \leftarrow T \cup \{c\}$
7: $\quad\quad \mathsf{wit}[c] \leftarrow v$
8: **return** $T$

▶ **Lemma 14** (⋆). *Let $T$ be the committee computed by Algorithm 1. Then $d(\mathsf{wit}[c], \mathsf{wit}[c']) > 2\sigma$ for any two distinct $c, c' \in T$.*

▶ **Lemma 15.** *If $\sigma \geq \sigma^*$, then Algorithm 1 outputs a size $k$ committee $T$ with $\sigma_f(T) \leq 3\sigma$.*

[3] We can check this condition by iterating over all failing sets of size $f$ and computing an optimal replacement set in each case.

**Proof.** The condition of the outer while loop of Algorithm 1 guarantees that $\delta(T, J) \leq 3\sigma$ for all $J \subseteq C$ of size at most $f$, which implies $\sigma_f(T) \leq 3\sigma$. To prove $|T| \leq k$, suppose $T = \{c_1, \ldots, c_r\}$. By Lemma 14, the pairwise distances between the voters $\mathsf{wit}[c_1], \ldots, \mathsf{wit}[c_r]$ are all larger than $2\sigma$ and thus larger than $2\sigma^*$ (as $\sigma \geq \sigma^*$ by our assumption). Now consider a committee $T^* \subseteq C$ of size $k$ satisfying $\sigma_f(T^*) = \sigma^*$. For each $\mathsf{wit}[c_i]$, there exists $c_i^* \in T^*$ such that $d(\mathsf{wit}[c_i], c_i^*) \leq \sigma^*$. Observe that $c_1^*, \ldots, c_r^*$ are all distinct. Indeed, if $c_i^* = c_j^*$ and $i \neq j$, then by the triangle inequality,

$$d(\mathsf{wit}[c_i], \mathsf{wit}[c_j]) \leq d(\mathsf{wit}[c_i], c_i^*) + d(\mathsf{wit}[c_j], c_j^*) \leq 2\sigma^*,$$

contradicting the fact that $d(\mathsf{wit}[c_i], \mathsf{wit}[c_j]) > 2\sigma^*$. Since $|T^*| = k$ and $c_1^*, \ldots, c_r^* \in T^*$, we have $r \leq k$, which completes the proof. ◀

Using these two lemmas, we can compute a 3-approximate solution using Algorithm 1 as follows. First, we compute $\sigma_f(C)$ in $O(nm^{f+1})$ time by enumerating all failing sets $J \subseteq C$ of size at most $f$. For every voter $v \in V$ and every candidate $c \in C$ such that $d(v, c) \geq \sigma_f(C)$, we run Algorithm 1 with $\sigma = d(v, c)$. Among all the committees returned of size at most $k$, we pick the one, say $T^*$, that minimizes $\sigma_f(T^*)$. To see that $\sigma_f(T^*) \leq 3\sigma^*$, note that $\sigma^*$ must be the distance between a voter and a candidate. Thus, there is one call of Algorithm 1 with $\sigma = \sigma^*$, which returns a committee $T \subseteq C$ of size at most $k$ such that $\sigma_f(T) \leq 3\sigma = 3\sigma^*$, by Lemma 15. We have $\sigma_f(T^*) \leq \sigma_f(T)$ by construction, which implies $\sigma_f(T^*) \leq 3\sigma^*$. In the full version, we show that each call of Algorithm 1 takes $O(nm^{2f+1})$ time. We need to call the algorithm $O(nm)$ times. Thus, we have the following result.

▶ **Theorem 16.** *We can find a* 3*-approximation for Optimal Fault-tolerant Committee in time* $O(n^2m^{2f+2})$*, assuming the fault-tolerance parameter* $f$ *is a constant.*

If we do not assume $f$ to be a constant, then the well-known "farthest first" greedy rule for adding candidates achieves a factor 5 approximation. Due to limited space, we describe the algorithm and its analysis in the appendix.

▶ **Lemma 17** (⋆)**.** *We can find a* 5*-approximation for Optimal Fault-Tolerant Committee in time* $\mathcal{O}(mnk)$*.*

All of the above approximations hold not just for $d$-dimensional Euclidean space, for any fixed $d$, but also for any metric space.

### 3.3.2 A bicriterion EPTAS

Finally, we design a bicriterion FPT approximation scheme with running time $f(\varepsilon) \cdot n^{\mathcal{O}(1)}$, which finds a size-$k$ committee whose fault-tolerance score for at least a $(1 - \varepsilon)$ fraction of the voters is within a factor of $(1 + \varepsilon)$ of the optimum. Formally, we say a committee $T$ is $(r, \rho)$*-good* if there exists a subset $V' \subseteq V$ of size at least $\rho n$ such that the $f$-tolerant score of $T$ with respect to only the voters in $V'$ is at most $r$. Then our approximation scheme can output a size-$k$ committee which is $((1 + \varepsilon)\sigma^*, 1 - \varepsilon)$-good. The core of our approximation scheme is the following (approximation) decision algorithm. The decision algorithm takes the problem instance and an additional number $r > 0$ as input. The output of the algorithm has two possibilities: it either **(i)** returns YES and gives a size-$k$ committee that is $((1 + \varepsilon)r, 1 - \varepsilon)$-good or **(ii)** simply returns NO. Importantly, the algorithm is guaranteed to give output **(i)** as long as $r \geq \sigma^*$. Note that this decision algorithm directly gives us the desired approximation scheme. Indeed, we can apply it with $r = d(v, c)$ for all $v \in V$ and $c \in C$. Let $r^*$ be the smallest $r$ that makes the algorithm give output **(i)**. The size-$k$

committee $T^*$ obtained when applying the algorithm with $r^*$ is $((1+\varepsilon)r^*, 1-\varepsilon)$-good. We have $r^* \leq \sigma^*$ because the algorithm must be applied with $r = \sigma^*$ at some point and it is guaranteed to give output **(i)** at that time. Thus, $T^*$ is $((1+\varepsilon)\sigma^*, 1-\varepsilon)$-good, as desired.

For simplicity of exposition, we describe our decision algorithm in two dimensions. By scaling, we may assume that the given number is $r = 1$. To solve the decision problem, our algorithm uses the shifting technique [18]. Let $h$ be an integer parameter to be determined later. For a pair of integers $i, j \in \mathbb{Z}$, let $\square_{i,j}$ denote the $h \times h$ square $[i, i+h] \times [j, j+h]$. A square $\square_{i,j}$ is *nonempty* if it contains at least one voter or candidate. We first compute the index set $I = \{(i,j) : \square_{i,j} \text{ is nonempty}\}$. This can be easily done in time $\mathcal{O}((n+m)h^2)$.

Consider a pair $(x,y) \in \{0,\ldots,h-1\}^2$. Let $L_{x,y}$ be the set of all integer pairs $(i,j)$ such that $i \pmod h \equiv x$ and $j \pmod h \equiv y$. We write $\widetilde{I}_{x,y} = \widetilde{I} \cap L_{x,y}$. For a voter $v \in V$ and a square $\square_{i,j}$, we say $v$ is a *boundary voter* for $\square_{i,j}$ if $v \notin [i+2, i+h-2] \times [j+2, j+h-2]$. Furthermore, we say $v$ *conflicts* with $(x,y)$ if $v$ is a boundary voter in $\square_{i,j}$ for some $(i,j) \in \widetilde{I}_{x,y}$.

▶ **Lemma 18** (⋆). *There exists a pair $(x,y) \in \{0,\ldots,h-1\}^2$ such that at most $\frac{4h-4}{h^2} \cdot |V|$ voters conflict with $(x,y)$.*

We fix a pair $(x,y) \in \{0,\ldots,h-1\}^2$ that conflicts with the minimum number of voters. For $(i,j) \in \widetilde{I}_{x,y}$, we define the set of (non-boundary) voters $V_{i,j} = \{v \in \square_{i,j} : v \in [i+2, i+h-2] \times [j+2, j+h-2]\}$, and the set of candidates $C_{i,j} = \{c \in C : c \in \square_{i,j}\}$. Note that for $(i,j) \in \widetilde{I}_{x,y}$, the $C_{i,j}$'s are disjoint and form a partition of $C$. Next, we show an important lemma which allows our algorithm to divide our problem into smaller subproblems, solve them individually, and combine the solutions to solve the overall problem.

▶ **Lemma 19** (⋆). *Let $V_1, V_2, \ldots, V_s$ be subsets of $V$ and let $T_1, T_2, \ldots, T_s$ be pairwise disjoint subsets of $C$ such that $T_i$ is a fault-tolerant committee for $V_i$ with $\sigma_f(T_i) = \sigma$. Then, $T = \bigcup_{i=1}^{s} T_i$ is a fault-tolerant committee of $\bigcup_{i=1}^{s} V_i$ with $\sigma_f(T) = \sigma$.*

Consider a pair $(i,j) \in \widetilde{I}_{x,y}$. Let $\overline{T}_{i,j}$ be a smallest fault-tolerant committee for $V_{i,j}$ with $\sigma_f(\overline{T}_{i,j}) \leq 1$. We observe that any inclusion-minimal fault-tolerant committee $T_{i,j}$ for $V_{i,j}$ satisfies $T_{i,j} \subseteq C_{i,j}$. This is because any candidate outside $C_{i,j}$ has distance more than $1 + 6/h$ to any voter in $V_{i,j}$ (for a large enough value of $h$). In the next section we will show how to compute a fault-tolerant committee $T_{i,j} \subseteq C_{i,j}$ for $V_{i,j}$ such that $|T_{i,j}| \leq |\overline{T}_{i,j}|$ and $\sigma_f(T_{i,j}) \leq 1 + 6/h$ in $h^{O(h^4)} n^{O(1)}$ time. Assuming we can compute the above-mentioned committee $T_{i,j}$, our overall algorithm is as follows:

1. Fix a pair $(x,y) \in \{0,\ldots,h-1\}^2$ conflicting with the minimum number of voters, and set $h$ to be the smallest integer such that $(4h-4)/h^2 \leq \varepsilon$ and $6/h \leq \varepsilon$.
2. For each pair $(i,j) \in \widetilde{I}_{x,y}$, compute $T_{i,j} \subseteq C_{i,j}$.
3. Let $T = \bigcup_{(i,j) \in \widetilde{I}_{x,y}} T_{i,j}$. If $|T| \leq k$, return YES (along with $T$); otherwise, return NO.

Let $V' = \bigcup_{(i,j) \in \widetilde{I}_{x,y}} V_{i,j}$. Since the $C_{i,j}$'s are disjoint, using Lemma 19, we conclude that $T$ is a fault-tolerant committee for $V'$. Furthermore, from our choice of $(x,y)$, we have $|V'| \geq (1-\varepsilon)n$. It is easy to show that the $f$-tolerant score of $T$ with respect to the voters in $V'$ is at most $1+\varepsilon$, and in addition, if $\sigma^* \geq 1$, we have $|T| \leq k$; we give a formal argument in the full version of our paper. This proves correctness of our decision algorithm. The overall algorithm takes $(1/\varepsilon)^{O(1/\varepsilon^4)}(m+n)^{O(1)}$ time. We note that the algorithm can be directly generalized to the $d$-dimensional case with running time $(1/\varepsilon)^{O(1/\varepsilon^{2d})}(m+n)^{O(1)}$. Therefore, we have the following result.

▶ **Theorem 20.** *Given a $d$-dimensional Fault-Tolerant Committee Selection instance, we can compute a size-$k$ committee $T$ such that the $f$-tolerant score of $T$ with respect to at least $(1-\varepsilon)n$ voters is at most $(1+\varepsilon)\sigma^*$, where $\sigma^*$ is the optimal $f$-tolerant score of a size-$k$ committee (with respect to the entire set $V$). This algorithm runs in time $(1/\varepsilon)^{O(1/\varepsilon^{2d})}(m+n)^{O(1)}$.*

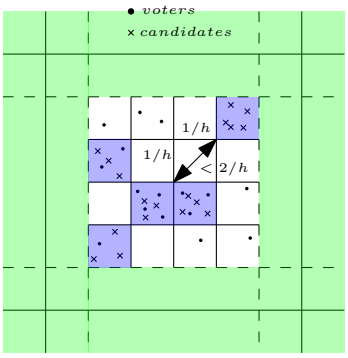



**Figure 2** The figure shows a cell in the shifted grid. The solid lines around the sides are the grid lines (and the region inside them is a cell). The shaded (green) region is the boundary region. Inside the boundary region, we divide the cell into $1/h \times 1/h$ smaller cells. The distance between any two points in a smaller cell is $< 2/h$. All candidates in smaller cells are identical (i.e., candidates in blue regions). In this example, since only five cells are nonempty, we have at most $2^5$ distinct failing sets.

### Algorithm to Compute $T_{i,j}$

We now present the most challenging piece of our algorithm: the computation of the $T_{i,j}$'s. Consider a box $\square_{i,j}$. Suppose there exists a fault-tolerant committee $T \subseteq C$ for $V_{i,j}$ with $\sigma_f(T) \leq 1$. Our task is to compute a fault-tolerant committee $T_{i,j} \subseteq C$ for $V_{i,j}$ such that $|T_{i,j}| \leq |T|$ and $\sigma_f(T_{i,j}) \leq 1 + 6/h$.

We divide $\square_{i,j}$ into $h^4$ smaller cells each with size $\frac{1}{h} \times \frac{1}{h}$, and we denote the set of these cells by $L = \{l_1, \ldots, l_{h^4}\}$. (See Figure 2.) Our algorithm is based on two key observations:
$(i)$ A committee with a candidate in every nonempty cell has $f$-tolerant score within a difference of at most $2/h$ from the optimum score. Since the number of cells is $h^4$, this implies that the size of a smallest approximately optimal committee is bounded by $h^4$ (formally shown in Lemma 21).
$(ii)$ All candidates in a cell can be treated as identical, causing only a loss of $2/h$ in the score. This implies that for any $T_{i,j}$, to approximately compute the $f$-tolerant score of $T_{i,j}$, we *only* need to consider the failing sets where either all or none of the candidates in a cell fail. Note that the number of such failing sets is at most $2^{O(h^4)}$ (formally shown in Lemma 22).

Using these two observations, at a high level, our algorithm goes through all committees of size at most $h^4$ (there are $h^{O(h^4)}$ of these as we can assume that each cell has at most $h^4$ candidates), approximately computes the $f$-tolerant score of each of these committees in time $2^{O(h^4)}$, and returns the smallest one with the desired score.

▶ **Lemma 21 (⋆).** *Let $T, T^* \subseteq C$ be fault-tolerant committees for $V_{i,j}$. If $|T^* \cap l_a| = 1$ for all $a \in [h^4]$ such that $C \cap l_a \neq \emptyset$, then $\sigma_f(T^*) - \sigma_f(T) \leq 2/h$.*

Based on the above observation, we solve the problem as follows. We enumerate all maps $\chi \colon L \to \{0, 1, \ldots, h^4\}$ where $\chi(l_a)$ is the number of candidates from $l_a$ in the committee. The total number of such maps is $h^{O(h^4)}$. For each feasible map, i.e., $\chi$ satisfying $\chi(l_a) \leq |C \cap l_a|$ for all $a \in [h^4]$, we construct a fault-tolerant committee $T^*_\chi$ for $V_{i,j}$ by picking (arbitrarily) $\chi(l_a)$ candidates in $C \cap l_a$ for all $a \in [h^4]$ and including them in $T^*_\chi$. For each constructed $T^*_\chi$, we compute a number $\widetilde{\sigma_f}(T^*_\chi)$ that approximates $\sigma_f(T^*_\chi)$ using the following lemma.

▶ **Lemma 22 (⋆).** *Given $T^*_\chi$, one can compute a number $\widetilde{\sigma_f}(T^*_\chi)$ in $2^{O(h^4)} n^{O(1)}$ time such that $|\widetilde{\sigma_f}(T^*_\chi) - \sigma_f(T^*_\chi)| \leq 2/h$.*

Finally, we let $T_{i,j}$ be the smallest among all committees $T^*_\chi$ satisfying $\widetilde{\sigma_f}(T^*_\chi) \leq 1 + 4/h$, and we return it as our solution. The running time of our algorithm is clearly $h^{O(h^4)}n^{O(1)}$. The following lemma shows that our algorithm is correct.

▶ **Lemma 23** (⋆). *We have $\sigma_f(T_{i,j}) \leq 1+6/h$. Furthermore, $|T_{i,j}| \leq |T|$ for any fault-tolerant committee $T$ for $V_{i,j}$ with $\sigma_f(T) \leq 1$.*

## References


1 Kenneth Arrow. *Advances in the spatial theory of voting*. Cambridge University Press, 1990.
2 Nadja Betzler, Arkadii Slinko, and Johannes Uhlmann. On the computation of fully proportional representation. *Journal of Artificial Intelligence Research*, 47:475–519, 2013.
3 Robert Bredereck, Piotr Faliszewski, Andrzej Kaczmarczyk, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Robustness among multiwinner voting rules. *Artificial Intelligence*, 290:103403, 2021.
4 Robert Bredereck, Andrzej Kaczmarczyk, and Rolf Niedermeier. On coalitional manipulation for multiwinner elections: Shortlisting. *Autonomous Agents and Multi-Agent Systems*, 35(2):38, 2021.
5 John R Chamberlin and Paul N Courant. Representative deliberations and representative decisions: Proportional representation and the borda rule. *American Political Science Review*, 77(3):718–733, 1983.
6 Shiva Chaudhuri, Naveen Garg, and Ramamoorthi Ravi. The p-neighbor k-center problem. *Information Processing Letters*, 65(3):131–134, 1998.
7 Otto A Davis, Melvin J Hinich, and Peter C Ordeshook. An expository development of a mathematical model of the electoral process. *American political science review*, 64(2):426–448, 1970.
8 Edith Elkind, Piotr Faliszewski, Jean-François Laslier, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. What do multiwinner voting rules do? an experiment over the two-dimensional euclidean domain. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31 of *AAAI'17*, pages 494–501, 2017.
9 Edith Elkind, Piotr Faliszewski, Piotr Skowron, and Arkadii Slinko. Properties of multiwinner voting rules. *Social Choice and Welfare*, 48(3):599–632, 2017.
10 Piotr Faliszewski, Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Llull and copeland voting computationally resist bribery and constructive control. *J. Artif. Intell. Res.*, 35:275–341, 2009. doi:10.1613/jair.2697.
11 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner voting: A new challenge for social choice theory. *Trends in computational social choice*, 74(2017):27–47, 2017.
12 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Committee scoring rules: Axiomatic characterization and hierarchy. *ACM Transactions on Economics and Computation (TEAC)*, 7(1):1–39, 2019.
13 Eric J Friedman, Vasilis Gkatzelis, Christos-Alexandros Psomas, and Scott Shenker. Fair and efficient memory sharing: Confronting free riders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1965–1972, July 2019.
14 Ashish Goel, Anilesh K Krishnaswamy, Sukolsak Sakshuwong, and Tanja Aitamurto. Knapsack voting for participatory budgeting. *ACM Transactions on Economics and Computation (TEAC)*, 7:1–27, 2019.
15 Arnaud Grivet Sébert, Nicolas Maudet, Patrice Perny, and Paolo Viappiani. Preference aggregation in the generalised unavailable candidate model. In *International Conference on Algorithmic Decision Theory*, pages 35–50. Springer, 2021.
16 Mohammadtaghi Hajiaghayi, Wei Hu, Jian Li, Shi Li, and Barna Saha. A constant factor approximation algorithm for fault-tolerant k-median. *ACM Transactions on Algorithms (TALG)*, 12(3):1–19, 2016.

17 Edith Hemaspaandra, Lane A Hemaspaandra, and Jörg Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5-6):255–285, 2007.

18 Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, pages 130–136, 1985.

19 Samir Khuller, Robert Pless, and Yoram J Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242(1-2):237–245, 2000.

20 Sven Oliver Krumke. On a generalization of the p-center problem. *Information processing letters*, 56(2):67–71, 1995.

21 Tyler Lu and Craig Boutilier. Budgeted social choice: From consensus to personalized decision making. In *Twenty-Second International Joint Conference on Artificial Intelligence*, pages 280–286, 2011.

22 Tyler Lu and Craig E Boutilier. The unavailable candidate model: a decision-theoretic view of social choice. In *Proceedings of the 11th ACM conference on Electronic commerce*, pages 263–274, 2010.

23 Sujaya Maiyya, Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. Database and distributed computing fundamentals for scalable, fault-tolerant, and consistent maintenance of blockchains. *Proceedings of the VLDB Endowment*, 11(12), 2018.

24 Neeldhara Misra and Chinmay Sonar. Robustness radius for chamberlin-courant on restricted domains. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 341–353. Springer, 2019.

25 Kamesh Munagala, Zeyu Shen, and Kangning Wang. Optimal algorithms for multiwinner elections and the chamberlin-courant rule. *EC '21: Proceedings of the 22nd ACM Conference on Economics and Computation*, pages 697–717, 2021.

26 Piotr Skowron, Piotr Faliszewski, and Jérôme Lang. Finding a collective set of items: From proportional multirepresentation to group recommendation. *Artificial Intelligence*, 241:191–216, 2016.

27 Piotr Skowron, Piotr Faliszewski, and Arkadii Slinko. Achieving fully proportional representation: Approximability results. *Artificial Intelligence*, 222:67–103, 2015.

28 Chinmay Sonar, Palash Dey, and Neeldhara Misra. On the complexity of winner verification and candidate winner for multiwinner voting rules. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 89–95, 2021.

29 Chinmay Sonar, Subhash Suri, and Jie Xue. Multiwinner elections under minimax chamberlin-courant rule in euclidean space. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 475–481. International Joint Conferences on Artificial Intelligence Organization, July 2022.

30 Chaitanya Swamy and David B Shmoys. Fault-tolerant facility location. *ACM Transactions on Algorithms (TALG)*, 4(4):1–27, 2008.

# Aggregating over Dominated Points by Sorting, Scanning, Zip and Flat Maps

**Jacek Sroka**
University of Warsaw, Poland

**Jerzy Tyszkiewicz**
University of Warsaw, Poland

## Abstract

Prefix aggregation operation (also called scan), and its particular case, prefix summation, is an important parallel primitive and enjoys a lot of attention in the research literature. It is also used in many algorithms as one of the steps.

Aggregation over dominated points in $\mathbb{R}^m$ is a multidimensional generalisation of prefix aggregation. It is also intensively researched, both as a parallel primitive and as a practical problem, encountered in computational geometry, spatial databases and data warehouses.

In this paper we show that, for a constant dimension $m$, aggregation over dominated points in $\mathbb{R}^m$ can be computed by $O(1)$ basic operations that include sorting the whole dataset, zipping sorted lists of elements, computing prefix aggregations of lists of elements and flat maps, which expand the data size from initial $n$ to $n \log^{m-1} n$.

Thereby we establish that prefix aggregation suffices to express aggregation over dominated points in more dimensions, even though the latter is a far-reaching generalisation of the former. Many problems known to be expressible by aggregation over dominated points become expressible by prefix aggregation, too.

We rely on a small set of primitive operations which guarantee an easy transfer to various distributed architectures and some desired properties of the implementation.

**2012 ACM Subject Classification** Theory of computation → Massively parallel algorithms; Theory of computation → Parallel computing models; Theory of computation → Database query processing and optimization (theory)

**Keywords and phrases** Aggregation over dominated points prefix sums sorting flat map range tree parallel algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.96

**Acknowledgements** We would like to thank Filip Murlak for encouragement and discussions on the topic. We also would like to thank anonymous referees for comments, which helped us improve the presentation of the paper.

## 1 Introduction

In this paper we derive a tight relation between two computing problems: prefix aggregation and aggregation over dominated points. We first describe the problems alone, in a framework which covers them both.

The input data is assumed to be a collection of tuples composed of sortable atomic elements. We are interested in aggregation in general. We assume the data consists of two sets: a set $D$ of data points and a set $Q$ of queries where for each query there is a subset $\hat{q} \subseteq D$ of data points it matches.

A nonempty set $A$ is the domain of weights. We are also given an associative and commutative function $\oplus : A \times A \to A$ with unit $e$, i.e., neutral element of $\oplus$.


Data points have weights in $A$, defined by a function $w : D \to A$. We use notation $\bigoplus_{d \in S} w(d)$ and the like for the application of $\oplus$ to a multiset of weights of elements $S \subseteq D$, the same way as $\Sigma$ is used as a generalisation of $+$ to a multiset of numbers.

The goal is to compute $\bigoplus_{d \in \hat{q}} w(d)$ for all $q \in Q$. The two problems we consider in the paper are specific cases of the above schematic outline. We assume $D$ and $Q$ to be large.

## 1.1 Prefix aggregation and its role as a parallel primitive

Prefix aggregation (also called scan) arises when data and queries are linearly ordered, say are both elements of $\mathbb{R}$, and $\hat{q} = \{d \in D \mid d < q\}$. Then we wish to compute $\bigoplus_{d<q} w(d)$. The eponymous case is when $D = Q$, so the aggregations are applied to all prefixes of the whole sequence of data points.

Concerning the importance of prefix aggregation as a computing primitive, it is well-known that, no matter what practical or theoretical parallel computation model is considered, sorting and scan are among the very first algorithms to be developed. The importance of the latter has even led to patents around the idea of including a prefix sum in the instruction set of a microprocessor [27], attempts of hardware implementations [16, 17] or adapting the actual algorithm to the architecture of the processor, sequential [14] or parallel [9]. Attempts of formal verification have also been undertaken [20, 10].

Scans were also researched as a primitive to implement parallel variants of many algorithms, including radix sort, quicksort, lexical analysis, polynomial evaluation, stream compaction, histograms and string comparison [4, 5], and also geometric partitioning algorithms [13].

## 1.2 Aggregation over dominated points

In this problem, data and queries are points in the $m$-dimensional space $\mathbb{R}^m$. For two such points we write $(x_1, \ldots, x_m) < (y_1, \ldots, y_m)$ when inequalities hold coordinate-wise, i.e., $x_1 < y_1, \ldots, x_m < y_m$. Then let $\hat{q} = \{d \in D | d < q\}$.

The result of the algorithm should therefore consist of $\bigoplus_{d<q} w(d)$ for each query point $q$, which is the result of applying $\oplus$ to the set of all weights of points in $D$ which are *dominated* by $q$, i.e., coordinate-wise smaller than $q$.

Aggregation over dominated points can obviously be considered as a multidimensional generalisation of 1-dimensional prefix aggregation. Note however that typically prefix aggregation uses an associative operation $\oplus$ with unit, while in the multidimensional setting we also require it to be commutative. The reason is that prefix aggregation has a natural order in which the elements are aggregated. In multidimensional setting there is no such natural order and hence, for the sake of producing a deterministic result, we require commutativity.

Aggregation over dominated points, referred to as *general prefix computations*, has been shown to be a parallel primitive which allows expressing many computational problems [22]. This approach has been subsequently extended to a general parallel computation model called *Broadcast with Selective Reduction* PRAM (BSR for short). The multiple criteria variant of BSR, introduced by Akl and Stojmenović [2, 3] after a number of earlier papers about single criterion BSR, is the one whose only parallel primitive is aggregation over dominated points. Many computational problems have been then shown to have constant-round BSR algorithms including: counting intersections of isothetic line segments, vertical segment visibility, maximal elements in $m$ dimensions, ECDF searching, 2-set dominance counting and rectangle containment in $m$ dimensions, rectangle enclosure and intersection counting in $m$ dimensions [2], all nearest smaller values [28], all nearest neighbours and furthest pairs of points in a plane in $L_1$ metric, the all nearest foreign neighbours in $L_1$ metric and the all furthest foreign pairs of points in the plane in $L_1$ metric [18]. All of them therefore can be expressed by aggregation over dominated points.

Other applications of this primitive include calculating Empirical Cumulative Distribution Functions (ECDFs) in statistics, which are required in multivariate Kolmogorov-Smirnov, Cramér-von Mises and Anderson-Darling statistical tests [15]. The problem is also intensively studied, under the name range-aggregate queries by the spatial database community [1, 26, 21] and data warehouse community [11]. However, in this area one typically does not assume queries to be known in advance, so the focus is rather on storing data points in a data structure which allows efficient querying.

## 1.3 Our contribution

We prove the following.

▶ **Theorem 1.** *Aggregation over dominated points in $\mathbb{R}^m$, where $m$ is constant, can be computed in $O(1)$ basic operations: sorting of lists of tuples, zipping and computing prefix aggregations as well as flat maps over such lists. By using flat lists, our algorithm expands the initial size of the input data from $n$ to $O(n \log^{m-1} n)$ tuples.*

Our work thus creates a direct link between so far separate parallel primitives. In this respect, we follow the paradigm presented by Blelloch [4, 5], who has shown that many computational problems can be expressed by prefix aggregation. Our result does not add just one more such problem, but, by transitivity, all problems which have been previously shown to be expressible by aggregation over dominated points. An interesting theoretical conclusion can be drawn, that prefix aggregation suffices to express aggregation over dominated points, i.e., its own multidimensional generalisation.

Seen from another prespective, our work significantly simplifies the algorithm from our earlier paper Sroka et al. [23]. It presents a constant-rounds algorithms for solving the counting variant of the problem, written for MapReduce and designed to be *minimal* in the sense proposed by Tao et al. [25], which guarantees it evenly distributes computation among worker nodes. It distributes range trees explicitly, and we borrow the method to do so from that paper. However, it uses recursion to deal with consecutive dimensions and minimal group-by method from [25] to aggregate the counts. We regard it as a new contribution that all data processing tasks of this algorithm are replaced by invocations of a very few simple primitives of well-understood behaviours, and that this result neatly connects two computational tasks, each one with its own history of research of algorithms it can express.

This switch from a particular parallel model to high-level parallel primitives makes the algorithm simpler to understand, reducing the number and level of details which must be taken care of. Another benefit is the fact that the algorithm now avoids any direct references to the mechanisms of the parallel hardware it is running on, like processors, messages, shared resources, etc. It requires exactly those, which are used by the underlying implementations of of the primitives we rely on. Among them, prefix aggregation is the only one, which allows combining values from an unbounded number of data elements together. However, the role of flat maps is also crucial, because they distribute certain computations, the results of which prefix aggregation later reduces. Finally, scans allow distributing many algorithms which in the centralised setting are based on sorting and iterating over data. Such approach has many advantages similar to those pointed out in [25], e.g., the resulting algorithms have strong guarantees concerning the way they distribute work and load. The ideas we present this paper can be viewed as generalisation of such approach to multidimensional setting the same way as range tree generalises binary search tree.

There is also an algorithm by Yufei Tao [24][Theorem 5] that uses an entirely different idea. It is directly tailored for the MPC model and takes care of reducing the maximal amount of communication between processors. It is based on partitioning space into fragments,

recursively solving problem over them and finally aggregating partial results. It achieves the optimal load $m^{O(m)}N/p$ with $p$ processors. As far as we understand, neither of them can be adapted to use prefix aggregation as its main mechanism.

# 2 Primitives

## 2.1 Data model

We assume the data to be stored in immutable but ordered lists. We are going to transform lists into new lists. Such approach and immutability is typical for distributed architectures, while ordering can be achieved by imposing some order on nodes in the cluster and distributing values such that successive nodes have increasing elements. Initial ordering of the input data is arbitrary, but we assume that input values are equipped with some numerical IDs that define it and can be used to break ties if needed.

The initial list of data points (vectors in $\mathbb{R}^m$) is going to be referred to as $D$ and the list of queries as $Q$. The weights of data points are represented by weight function $w : D \to A$. To make the exposition simpler, we assume that weights are defined for queries, too, and $w(q) = e$ for $q \in Q$, so that they do not interfere with aggregation.

## 2.2 Primitives and macros

In this section we postulate primitive operations that are used to express our algorithms as well we define some convenient macros that combine them.

▶ **Definition 2** (Sort). *For $x = [x_0, \ldots, x_n]$ and some linear order relation $\preccurlyeq \subseteq X \times X$*

$$\mathbf{sort}(x, \preccurlyeq) = [x_{i_1}, \ldots, x_{i_n}],$$

*where the multisets $\{\!\{x_{i_1}, \ldots x_{i_n}\}\!\}$ and $\{\!\{x_1, \ldots x_n\}\!\}$ are equal, and $x_{i_j} \preccurlyeq x_{i_{j+1}}$ for all $j$.* ⌟

It is known that radix sort and quicksort are expressible by prefix aggregation [4, 5], hence we could theoretically eliminate sorting from the list of primitives we rely on.

▶ **Definition 3** (FlatMap). *For $x = [x_0, \ldots, x_n]$ and $f : X \to [Y]$, which applied to an element produces a list of elements as the result:*

$$\mathbf{flatmap}(x, f) = f(x_0) \,\&\, \ldots \,\&\, f(x_n),$$

*where $\&$ is list concatenation.* ⌟

For convenience we define map, which expects $f : X \to Y$ to produce single elements, by using FlatMap and composing $f$ with list constructor $list()$:

$$\mathbf{map}(x, f) := \mathbf{flatmap}(x, list() \circ f),$$

so that

$$\mathbf{map}([x_1, \ldots, x_n], f) = [f(x_1), \ldots, f(x_n)]$$

Zip is an operation which takes two (or more) lists of equal length and combines them into a single list of tuples, created from elements at the same positions.

▶ **Definition 4** (Zip). *For lists $x^1 = [x^1_1, \ldots, x^1_n], \ldots, x^k = [x^k_1, \ldots, x^k_n]$:*

$$\mathbf{zip}(x^1, \ldots, x^k) = [(x^1_1, \ldots, x^k_1), \ldots, (x^1_n, \ldots, x^k_n)].$$ ⌟

As usually immediately after zip we want to do something with those tuples, we define macros, which map or flatmap a provided function $f : X^k \to Y$ or $f : X^k \to [Y]$ on the tuples immediately:

$$\mathbf{mapzip}(x^1, \ldots, x^k, f) := \mathbf{map}(\mathbf{zip}(x^1, \ldots, x^k), f),$$

$$\mathbf{flatmapzip}(x^1, \ldots, x^k, f) := \mathbf{flatmap}(\mathbf{zip}(x^1, \ldots, x^k), f).$$

We define three variants of prefix aggregation of a list $[a_1, \ldots, a_n]$ of elements of $A$, known from literature.

▶ **Definition 5** (Scan). *For an aggregation operation $\oplus : A \times A \to A$, its natural element $e$ and a list $[a_1, \ldots, a_n]$ of elements of $A$:*

$$\mathbf{scan}([a_1, \ldots, a_n], \oplus) = [a_1, a_1 \oplus a_2, \ldots, a_1 \oplus \ldots \oplus a_n],$$

$$\mathbf{scan}^-([a_1, \ldots, a_n], \oplus) = [e, a_1, \ldots, a_1 \oplus \ldots \oplus a_{n-1}].$$ ⌟

As we need $\mathbf{scan}^-$ only for aggregating nondecreasing lists of numerical values with $\oplus = \max$ and $e = -\infty$ we define a macro: $\mathbf{shift}([x_1, \ldots, x_n]) := \mathbf{scan}^-([x_1, \ldots, x_n], \max) = [-\infty, x_1, \ldots, x_{n-1}]$, which is indeed a right-shift of its input if $[x_1, \ldots, x_n]$ is nondecreasing.

Another useful macro for lists of numerical values with $\oplus = \max$ and $e = -\infty$ is the following: $\mathbf{broadcastmax}([x_1, \ldots, x_n]) := \mathbf{scan}(\mathbf{sort}([x_1, \ldots, x_n], \geq), \max) = [x, \ldots, x]$ where $x = \max(x_1, \ldots, x_n)$. This macro indeed broadcasts the maximal value in a list to all positions. By zipping this list with another list we assure that the maximal value can be used for local processing of the latter, by **map**.

▶ **Definition 6** (Segmented scan). *For an aggregation operation $\oplus : A \times A \to A$ and two lists $[a_1, \ldots, a_n]$ of elements of $A$ and $[t_1, \ldots, t_n]$ of elements of some other set $T$, where the latter list is sorted:*

$$\mathbf{sscan}([a_1, \ldots, a_n], [t_1, \ldots, t_n], \oplus) = [\bigoplus_{\substack{i \leq 1 \\ t_i = t_1}} a_i, \bigoplus_{\substack{i \leq 2 \\ t_i = t_2}} a_i, \ldots, \bigoplus_{\substack{i \leq n \\ t_i = t_n}} a_i].$$

This means, that we essentially decompose the first argument list into maximal segments over which the corresponding elements of the second list remain identical, and then compute $\mathbf{scan}(s, \oplus)$ for each segment $s$ separately, e.g., $\mathbf{sscan}([1, 2, 3, 4, 5, 6], [0, 0, 1, 1, 1, 2], +) = [1, 3, 3, 7, 12, 6]$.

Segmented scan can be expressed by standard scan (see [4]), but is typically designed and implemented independently, which gives a chance for better performance, in particular on complex, constrained architectures, such as GPU [8]. It can also be implemented as a data oblivious algorithm, whose memory access pattern is independent of the actual data being processed [19]. Note that scan is our only operation for combining unbounded number of elements, here by aggregation into a single value.

The primitives described in this section essentially define our model of hardware the algorithm is running on.

## 3 Checking dominance by polylogarithmic data expansion

In this section we present an important tool we need in our algorithm. It allows us to distribute the process of checking dominance relations later on. It can be viewed as a method to distribute a range tree that allows to query multidimensional data. It derives from the paper [23].

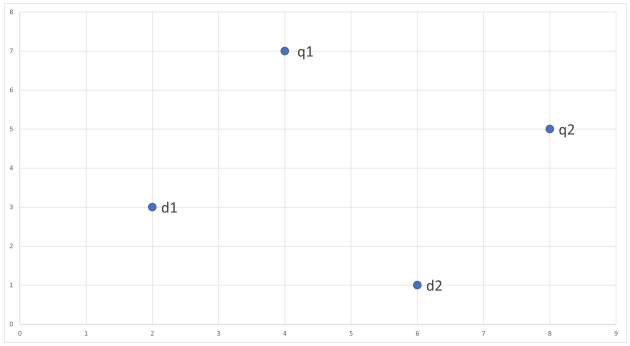



**Figure 1** Geometric visualization of the example data and queries. Note that the aggregation output in this case should be $w(d1)$ for $q1$ and $w(d1) \oplus w(d2)$ for $q2$.

We consider natural numbers written in binary notation, padded to some fixed length with leading 0's. Let $x$ be a bitstring. Then let $P0(x)$ be the set of all bitstrings $v$ such that $v0$ is a prefix of $x$, including the empty prefix, should $x$ start with a 0, e.g., $P0(01010) = \{0101, 01, \varepsilon\}$. Similarly, let $P1(x)$ be the set of all bitstrings $v$, such that $v1$ is a prefix of $x$.

▶ **Lemma 7.** *Suppose $x$ and $y$ are natural numbers represented as bitstrings of equal length, perhaps with leading 0's.*

*If $x < y$ then $P0(x) \cap P1(y)$ has exactly one element, and if $x \geq y$ then $P0(x) \cap P1(y) = \emptyset$.*

**Proof.** $x < y$ iff their longest common prefix is followed by 0 in $x$ and by 1 in $y$. Hence $P0(x) \cap P1(y)$ is nonempty iff $x < y$, which takes care of the $x \geq y$ part.

To rule out the possibility that $x < y$ and $P0(x) \cap P1(y)$ has more than 1 element, it is enough to observe that the longest element in $P0(x) \cap P1(y)$ is at the same time the shortest one because it has to be followed by different symbols in $x$ and $y$. ◀

Let $(x_1, \ldots, x_n)$ be a tuple of bitstrings. Define $P0((x_1, \ldots, x_n)) = P0(x_1) \times \ldots \times P0(x_n)$, and similarly $P1((x_1, \ldots, x_n)) = P1(x_1) \times \ldots \times P1(x_n)$.

▶ **Lemma 8.** *Let $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ be two tuples of natural numbers encoded as bitstrings, coordinate-wise of equal lengths.*

*If $\vec{x}$ is coordinate-wise smaller than $\vec{y}$ then $P0(\vec{x}) \cap P1(\vec{y})$ is a singleton, and otherwise it is empty.*

**Proof.** Follows from Lemma 7. ◀

## 4 Algorithm

We present the algorithm in several groups of numbered instructions and for each add explanations. Each such group is followed by an example of its action on a very small 2-dimensional dataset with $D = [(2,3)_{d1}, (6,1)_{d2}]$ and $Q = [(4,7)_{q1}, (8,5)_{q2}]$, which is intended to help with the explanation of the algorithm. Value subscripts indicate their IDs, whose order is $d1, q1, d2, q2$. See Figure 1 for a geometrical visualization.

The algorithm works on data and query points together, so first the union $DQ = D \cup Q$ of those two lists is created.

| | |
|---|---|
| 0 | $DQ = \textbf{flatmapzip}(D, Q, \lambda x, y.[x, y])$ |

| | |
|---|---|
| 0 | $DQ = [(2,3)_{d1}, (4,7)_{q1}, (6,1)_{d2}, (8,5)_{q2}]$ |

The next group of instructions is used to compute, for each dimension, the rank of each tuple's coordinate in that dimension and the total number of unique coordinates.

Let $less : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be defined as

$$less(a,b) = \begin{cases} 1 & \text{if } a < b \\ 0 & \text{if } a \geq b \end{cases}.$$

For each dimension $i = 1, \ldots, m$ we add the following instructions.

| | |
|---|---|
| $5i-4$ | $S_i = \mathbf{map}(DQ, \lambda x.x[i])$ |
| $5i-3$ | $T_i = \mathbf{sort}(S_i, \leq)$ |
| $5i-2$ | $W_i = \mathbf{shift}(T_i)$ |
| $5i-1$ | $R_i = \mathbf{scan}(\mathbf{mapzip}(W_i, T_i, less), +)$ |
| $5i$ | $U_i = \mathbf{broadcastmax}(R_i)$ |

| | for $i = 1$: | | for $i = 2$: |
|---|---|---|---|
| 1 | $S_1 = [2_{d1}, 4_{q1}, 6_{d2}, 8_{q2}]$ | 6 | $S_2 = [3_{d1}, 7_{q1}, 1_{d2}, 5_{q2}]$ |
| 2 | $T_1 = [2_{d1}, 4_{q1}, 6_{d2}, 8_{q2}]$ | 7 | $T_2 = [1_{d2}, 3_{d1}, 5_{q2}, 7_{q1}]$ |
| 3 | $W_1 = [-\infty, 2_{d1}, 4_{q1}, 6_{d2}]$ | 8 | $W_2 = [-\infty, 1_{d2}, 3_{d1}, 5_{q2}]$ |
| 4 | $R_1 = [1,2,3,4]$ | 9 | $R_2 = [1,2,3,4]$ |
| 5 | $U_1 = [4,4,4,4]$ | 10 | $U_2 = [4,4,4,4]$ |

Line $5i-4$ extracts the sequence of $i$-th coordinates of vectors from $DQ$, which is then sorted in line $5i-3$, so that it can be shifted by one position to the right in line $5i-2$. Then $less$ in line $5i-1$ essentially compares each element of $T_i$ with its predecessor and produces a list of $1s$ and $0s$, with 1 on positions with a difference and 0 otherwise. Therefore prefix sum of that sequence computes ranks of the elements of $T_i$. In particular, on the last index there will be total number of unique elements. We need a list with this value present at every position. It is computed in line $5i$ with **broadcastmax**.

Now we transform each rank into its binary representation of fixed length which can be viewed as coordinates of that value in a binary search tree. Let $bin(n,k)$ for $n \leq k$ be defined as a binary expansion of $n-1$ using exactly $\lceil \log k \rceil$ binary digits, i.e., with leading zeros if necessary.

Again for each dimension $i = 1, \ldots, m$ we add the following instructions.

| | |
|---|---|
| $5m+i$ | $BR_i = \mathbf{sort}(\mathbf{mapzip}(R_i, U_i, bin), ID)$ |

| | for $i = 1$: | | for $i = 2$: |
|---|---|---|---|
| 11 | $BR_1 = [00_{d1}, 01_{q1}, 10_{d2}, 11_{q2}]$ | 12 | $BR_2 = [01_{d1}, 11_{q1}, 00_{d2}, 10_{q2}]$ |

The instructions in lines $5m+1, \ldots, 6m$ transform the original data in each dimension to the rank space, where values are replaced by their ranks within the whole dataset, written in binary. Ranks isomorphically preserve all inequalities of the original real values, and hence preserve the results of aggregations to be computed, too.

Let $P0(x)$ and $P1(x)$ be as in Lemma 8. Furthermore, let $P$ be a function from $m$-tuples of binary strings to sets of $m$-tuples of binary strings, defined as follows:

$$P(b_1, \ldots, b_m) = \begin{cases} P0(b_1, \ldots, b_m) & \text{if } (b_1, \ldots, b_m) \text{ is a data point;} \\ P1(b_1, \ldots, b_m) & \text{if } (b_1, \ldots, b_m) \text{ is a query point.} \end{cases}$$

| | |
|---|---|
| $6m+1$ | $EDQ = \mathbf{flatmapzip}(BR_1, \ldots, BR_m, P)$ |

| | |
|---|---|
| 13 | $EDQ = [(\varepsilon,\varepsilon)_{d1}, (0,\varepsilon)_{d1}, (0,\varepsilon)_{q1}, (0,1)_{q1}, (1,\varepsilon)_{d2}, (1,0)_{d2}, (\varepsilon,\varepsilon)_{q2}, (1,\varepsilon)_{q2}]$ |

This results in a new, expanded sequence $EDQ$ consisting of tuples of bitstrings. We assume they inherit ID and weights from their originating data and query points. This sequence will contain many duplicated tuples, but with different IDs. This operation increases the total size of data from $O(n)$ to $O(n \log^m n)$.

Let $\leq_{lex}$ be doubly lexicographic ordering relation on tuples from $EDQ$: lexicographic for bitstrings in each coordinate and lexicographic between coordinates, with the (crucial) additional requirement, that in case of a tie of two tuples data points precede queries.

| | |
|---|---|
| $6m+2$ | $SEDQ = \mathbf{sort}(EDQ, \leq_{lex})$ |

| | |
|---|---|
| 14 | $SEDQ = [(\varepsilon,\varepsilon)_{d1}, (\varepsilon,\varepsilon)_{q2}, (0,\varepsilon)_{d1}, (0,\varepsilon)_{q1}, (0,1)_{q1}, (1,\varepsilon)_{d2}, (1,\varepsilon)_{q2}, (1,0)_{d2}]$ |

$SEDQ$ is composed of segments consisting of duplicates, differing only by IDs and weights. Data points come before queries among each segment of equal values. At this point we have prepared all data we need, it remains to perform several steps of aggregations.

| | |
|---|---|
| $6m+3$ | $A_1 = \mathbf{sscan}(\mathbf{map}(SEDQ, \lambda x.w(x)), SEDQ, \oplus)$ |

| | |
|---|---|
| 15 | $A_1 = [w(d1)_{d1}, w(d1)_{q2}, w(d1)_{d1}, w(d_1)_{q1}, e_{q1}, w(d2)_{d2}, w(d2)_{q2}, w(d2)_{d2}]$ |

In line $6m+3$ we compute segmented prefix sums of the weights of the sorted data and queries. All bitstring coordinates serve as tags for segmentation. This way a list is created, where each query point $q$ corresponds (by position on the list) to the aggregation of all weights of data points which yielded the same tuple of bitstrings. By Lemma 8, each such identical tuple originates from a data point $d$ such that $d < q$, and, moreover, for each fixed tuple $t$ derived from $q$ there is one-to-one correspondence between such tuples and data points which also produced $t$ and are (therefore) dominated by $q$.

Queries have been assigned neutral weight, so they do not interfere with the scan. However, some aggregation happens at the positions of data points, too.

Let $\leq_{ID}$ be ordering relation on tuples which compares their associated $ID$ values.

| | |
|---|---|
| $6m+4$ | $A_2 = \mathbf{sort}(A_1, \geq_{ID})$ |
| $6m+5$ | $A_3 = \mathbf{sscan}(A_2, \mathbf{map}(A_1, \lambda x.ID(x)), \oplus)$ |

16 $A_2 = [w(d2)_{q2}, w(d1)_{q2}, w(d2)_{d2}, w(d2)_{d2}, e_{q1}, w(d1)_{q1}, w(d1)_{d1}, w(d1)_{d1}]$
17 $A_3 = [w(d2)_{q2}, (w(d2) \oplus w(d1))_{q2}, w(d2)_{d2}, (w(d2) \oplus w(d2))_{d2},$
$e_{q1}, w(d1)_{q1}, w(d1)_{d1}, (w(d1) \oplus w(d1))_{d1}]$

Line $6m + 4$ is a sort of partial aggregations by $ID$ in nonincreasing sequence, which creates a continuous segment of aggregation values corresponding to each query. There are also separate segments of data points, which are irrelevant now.

After that in line $6m + 5$ we compute segmented prefix sums of the already partially aggregated weights of the sorted data and queries, whereby their $ID$ values serve as tags for segmentation. This way partial aggregations for each query are further aggregated, so that the total aggregation of each query $q$ corresponds, by position, to the last tuple which originated from $q$. This total aggregation for $q$ comes from all $d$ such that $P(q) \times P(d) \neq \emptyset$ (those sets are created in line $6m + 1$). By Lemma 8, $\{d|P(q) \cap P(d) \neq \emptyset\} = \{d|d < q\}$, and each such $d$ in the r.h.s. is witnessed by exactly one element of $P(q) \cap P(d)$.

At this point the aggregation of each query is already computed, but the data contains many partial aggregations for the same query, too. Therefore the last task is to distribute the total aggregation of each query to all tuples with the same ID.

This can be significantly simplified if $A$ is linearly ordered and $a \oplus b \geq a$ for all $a, b \in A$. In this case for each query we need the maximum aggregation among all partial ones. Therefore a segmented variant of **broadcastmax** with ID defining segmentation would do that. Otherwise the method to achieve the goal is more complex and described below.

Let $neutral\_if\_eq(id_1, id_2, a) = \begin{cases} e & \text{if } id_1 = id_2 \\ a & \text{if } id_1 \neq id_2 \end{cases}$.

$6m + 6$ $A_4 = \mathbf{sort}(A_3, \leq_{ID})$
$6m + 7$ $H = \mathbf{shift}(\mathbf{map}(A_4, \lambda x.ID(x)))$
$6m + 8$ $A_5 = \mathbf{mapzip}(A_4, H, \mathbf{map}(A_4, \lambda x.ID(x)), neutral\_if\_eq)$
$6m + 9$ $Out = \mathbf{sscan}(A_5, \mathbf{map}(\lambda x.ID(x), A_5), \oplus)$

18 $A_4 = [(w(d1) \oplus w(d1))_{d1}, w(d1)_{d1}, w(d1)_{q1}, e_{q1}, (w(d2) \oplus w(d2))_{d2}, w(d2)_{d2},$
$(w(d2) \oplus w(d1))_{q2}, w(d2)_{q2}]$
19 $H = [-\infty, d1, d1, q1, q1, d2, d2, q2]$
20 $A_5 = [(w(d1) \oplus w(d1))_{d1}, e_{d1}, w(d1)_{q1}, e_{q1}, (w(d2) \oplus w(d2))_{d2}, e_{d2},$
$(w(d2) \oplus w(d1))_{q2}, e_{q2}]$
21 $Out = [(w(d1) \oplus w(d1))_{d1}, (w(d1) \oplus w(d1))_{d1}, w(d1)_{q1}, w(d1)_{q1}, (w(d2) \oplus w(d2))_{d2},$
$(w(d2) \oplus w(d2))_{d2}, (w(d2) \oplus w(d1))_{q2}, (w(d2) \oplus w(d1))_{q2}]$

In line $6m+6$ we reverse the sort order of $A_4$. Now the complete aggregations come at the beginning of each segment, and, moreover, the $ID$ values are nondecreasing, hence we can shift them in line $6m+7$. Line $6m+8$ resets all computed weights to the neutral $e$, except at the beginning of each segment, where the complete aggregation is present. Finally, segmented scan in line $6m + 9$ aggregates these values with neutral elements elsewhere, producing the desired output.

Now each query ID is accompanied by the aggregation of weights of its dominated points, which means that the desired output has been computed. In total it took $6m+9$ instructions and at most that many intermediate lists of data created. As it can be seen, the output of the example agrees with the output determined from the geometric presentation in Figure 1.

## 5 Improvement by one logarithm

It is possible to reduce the amount of data generated by a factor of $\log n$.

One of the coordinates (say: the last one) is chosen. It is not replaced by ranks and left in the form of real numbers. The remaining ones are replaced by ranks and prefixes are generated, exactly as in the basic algorithm, from both data points and queries. This results in a multiset of $\leq n \log^{m-1} n$ tuples with $m-1$ coordinates in the form of bitstrings and the last, $m$-th coordinate being real number. Identifiers are retained.

Now sort the data and queries $EDQ$ into $SEDQ$ (see line $6m+2$ and explanation thereof above) according to the doubly lexicographic order, with queries preceding data points in case of equality of all coordinates (including the $m$-th).

This results in segments of data elements with $m-1$ first coordinates equal, sorted according to the last coordinate within the segment. Then the remainder of the algorithm is executed exactly as in the basic version.

## 6 Relation to range trees and complexity

The algorithm we have presented above is derived from our earlier MapReduce algorithm Sroka et al. [23], and is indeed a parallelisation of the common sequential algorithm, based on range trees. The move to the rank space with ranks expressed as binary expansions is equivalent to speaking about elements in terms of their positions in a balanced binary tree, whose leaves hold the sorted data. The binary encodings then correspond to branches in the tree, and their prefixes to the positions where attached trees of smaller dimensions are located.

Our algorithm inherits its total data complexity of $O(n \log^{m-1} n)$ from the range tree algorithm. This distributed data structure is generated by flat maps, while the parallelisation is achieved by expressing operations on the tree in terms of sorting, zipping and prefix aggregation.

Time complexity of our algorithm depends very much on the underlying architecture and complexity of the primitive operations, but its analysis is pretty straightforward in each case, since it is a fixed length sequence of operations of very well known properties, applied to lists of data of sizes easy to determine. In particular, no matter what architecture it is executed on, if the implementations of primitives do the same total work as their sequential variants, then the whole algorithm will also have the total work of the sequential algorithm using range trees.

Indeed, in the sequential case the only nonlinear (and thus dominating) operation is sorting, and the size of the data is $O(n \log^{m-1} n)$. One linearithmic sort takes time $O(n \log^{m-1} n \cdot \log(n \log^{m-1} n)) = O(n \log^m n)$, which is equal to the worst case of the standard sequential implementation, calculated as creating the range tree with $n/2$ data points and then processing $n/2$ queries by this tree. There is a one logarithm better variant of Chazelle [6], which however works only for counting.

For the MPC model, it is known that for $\delta > 0$, sorting and scanning of $n$ values can be performed deterministically in a constant number of rounds using $n^\delta$ space per machine, $O(n)$ total space, and poly$(n)$ local computation, which follows directly from analogous bounds for MapReduce computation. The load can be made $O(N/p)$ [7]. This implies that our algorithm in dimension $m$ can be implemented deterministically in $O(m)$ rounds, with $n^\delta$ space per machine, $O(n \log^{m-1} n)$ total space and poly$(n)$ local computation, with load $O(N \log^{m-1} N/p)$.

By comparison, the algorithm by Hu et al. [12] achieves load $O(Np^{-1}\log^{m-1} p)$ with $p$ processors, which is better than what get in this paper. This is not surprising, since our reliance on high-level primitives gives us much less freedom in designing the computation mechanism and achieving low loads. In particular, the set of instructions we use does not permit shifting the computational effort into the local computation, which is the method to lower the loads. The algorithm of Tao [24] does it to an even higher extent, arriving at load $m^{O(m)}N/p$.

Our earlier algorithm from Sroka et al. [23] is quite similar, but at that time we did only tests of an implementation on MapReduce, for which it has been designed. The running times did not seem to scale linearly with the number of machines. Moreover, for its present form we can do complexity analysis, greatly simplified by known complxities of the primitives we use.

## 7 Summary and future research

In this paper we have presented algorithm for aggregating over dominated points in $\mathbb{R}^m$, where $m$ is constant. Our algorithm is based on a limited set of primitive operations: sorting, prefix aggregations, zip and flat maps. All those primitive operations are well studied and their efficient implementations exist for essentially all distributed architectures.

This proves that one-dimensional prefix aggregation allows expressing its own multidimensional generalisation. The latter problem has many practical applications, as well as it is known to be a parallel primitive, allowing to express in turn further problems. By transitivity, our result expresses all those problems in terms of the above mentioned primitive operations.

We consider our result to be primarily of theoretical interest at present, before experimental tests are conducted.

We believe that our algorithm may turn out to be quite practical. First of all, it is absolutely transparent and does not hide any significant computation steps. The local computation is on the level of individual tuples, only. No large collections of data need to be broadcasted to computation nodes and there is no limit on number of such nodes. Otherwise we use high-level primitives of very well understood algorithmic properties, and whose highly optimised implementations exist for virtually all hardware platforms. Also the organisation of the algorithm into a sequence of functional operations (without any branch) on immutable ordered lists is very convenient for implementation. Last but not least, the choice of primitives guarantees that the algorithm has several desired properties, e.g., it is minimal in the sense of [25].

On the other hand, the $\log^{m-1} n$ memory footprint will be a problem in larger dimensions.

Therefore an obvious item on the "further research" list is undertaking experiments with the algorithm on diverse parallel platforms.

## References

1 Pankaj K. Agarwal, Lars Arge, Sathish Govindarajan, Jun Yang, and Ke Yi. Efficient external memory structures for range-aggregate queries. *Computational Geometry*, 46(3):358–370, 2013. doi:10.1016/j.comgeo.2012.10.003.

2 Selim G. Akl and Ivan Stojmenović. Multiple criteria BSR: An implementation and applications to computational geometry problems. In *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, 1994.

3 Selim G. Akl and Ivan Stojmenović. Broadcasting with selective reduction: A powerful model of parallel computation. *Parallel and distributed computing handbook*, pages 192–222, 1996.

4 Guy E. Blelloch. Scans as primitive parallel operations. *IEEE Trans. Computers*, 38(11):1526–1538, 1989. doi:10.1109/12.42122.

5 Guy E. Blelloch. Prefix sums and their applications. In John H Reif, editor, *Synthesis of parallel algorithms*. Morgan Kaufmann Publishers Inc., 1993.

6 Bernard Chazelle. Lower bounds for orthogonal range searching II. the arithmetic model. *J. ACM*, 37(3):439–463, 1990. doi:10.1145/79147.79149.

7 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, volume 7074 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011. doi:10.1007/978-3-642-25591-5_39.

8 Mark Harris and Michael Garland. Optimizing parallel prefix operations for the Fermi architecture. In Wen mei W. Hwu, editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 29–38. Morgan Kaufmann, Boston, 2012. doi:10.1016/B978-0-12-385963-1.00003-4.

9 Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with CUDA. *GPU gems*, 3(39):851–876, 2007.

10 Ralf Hinze. An algebra of scans. In Dexter Kozen and Carron Shankland, editors, *Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings*, volume 3125 of *Lecture Notes in Computer Science*, pages 186–210. Springer, 2004. doi:10.1007/978-3-540-27764-4_11.

11 Seokjin Hong, Byoungho Song, and Sukho Lee. Efficient execution of range-aggregate queries in data warehouse environments. In Hideko S. Kunii, Sushil Jajodia, and Arne Sølvberg, editors, *Conceptual Modeling - ER 2001, 20th International Conference on Conceptual Modeling, Yokohama, Japan, November 27-30, 2001, Proceedings*, volume 2224 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2001. doi:10.1007/3-540-45581-7_23.

12 Xiao Hu, Ke Yi, and Yufei Tao. Output-optimal massively parallel algorithms for similarity joins. *ACM Trans. Database Syst.*, 44(2):6:1–6:36, 2019. doi:10.1145/3311967.

13 Y. Charlie Hu, Shang-Hua Teng, and S. Lennart Johnsson. A data-parallel implementation of the geometric partitioning algorithm. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1997*. SIAM, 1997.

14 Hung-Ming Lai and Jenq-Kuen Lee. Efficient support of the scan vector model for RISC-V vector extension. In *Workshop Proceedings of the 51st International Conference on Parallel Processing, ICPP Workshops 2022, Bordeaux, France, 29 August 2022 - 1 September 2022*, pages 15:1–15:8. ACM, 2022. doi:10.1145/3547276.3548518.

15 Nicolas Langrené and Xavier Warin. Fast multivariate empirical cumulative distribution function with connection to kernel density estimation. *Computational Statistics & Data Analysis*, 162:107267, 2021. doi:10.1016/j.csda.2021.107267.

16 Rong Lin, Koji Nakano, Stephan Olariu, Maria Cristina Pinotti, James L. Schwing, and Albert Y. Zomaya. Scalable hardware-algorithms for binary prefix sums. *IEEE Trans. Parallel Distributed Syst.*, 11(8):838–850, 2000. doi:10.1109/71.877941.

17 Rong Lin, Koji Nakano, Stephan Olariu, and Albert Y. Zomaya. An efficient parallel prefix sums architecture with domino logic. *IEEE Trans. Parallel Distributed Syst.*, 14(9):922–931, 2003. doi:10.1109/TPDS.2003.1233714.

18 Robert A. Melter and Ivan Stojmenovic. Constant time BSR solutions to $l_1$ metric and digital geometry problems. *J. Math. Imaging Vis.*, 5(2):119–127, 1995. doi:10.1007/BF01250524.

19 Vijaya Ramachandran and Elaine Shi. Data oblivious algorithms for multicores. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 373–384. ACM, 2021. doi:10.1145/3409964.3461783.

**20** Mohsen Safari and Marieke Huisman. Formal verification of parallel prefix sum and stream compaction algorithms in CUDA. *Theor. Comput. Sci.*, 912:81–98, 2022. doi:10.1016/j.tcs.2022.02.027.

**21** Yexuan Shi, Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Bolin Ding, and Lei Chen. Efficient approximate range aggregation over large-scale spatial data federation. *IEEE Trans. Knowl. Data Eng.*, 35(1):418–430, 2023. doi:10.1109/TKDE.2021.3084141.

**22** Frederick N. Springsteel and Ivan Stojmenovic. Parallel general prefix computations with geometric, algebraic, and other applications. *Int. J. Parallel Program.*, 18(6):485–503, 1989. doi:10.1007/BF01381719.

**23** Jacek Sroka, Artur Leśniewski, Miroslaw Kowaluk, Krzysztof Stencel, and Jerzy Tyszkiewicz. Towards minimal algorithms for big data analytics with spreadsheets. In Foto N. Afrati and Jacek Sroka, editors, *Proceedings of the 4th ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond, BeyondMR@SIGMOD 2017, Chicago, IL, USA, May 19, 2017*, pages 1:1–1:4. ACM, 2017. doi:10.1145/3070607.3075961.

**24** Yufei Tao. Massively parallel entity matching with linear classification in low dimensional space. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, volume 98 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICDT.2018.20.

**25** Yufei Tao, Wenqing Lin, and Xiaokui Xiao. Minimal MapReduce algorithms. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 529–540, New York, NY, USA, 2013. ACM. doi:10.1145/2463676.2463719.

**26** Yufei Tao and Dimitris Papadias. Range aggregate processing in spatial databases. *IEEE Trans. Knowl. Data Eng.*, 16(12):1555–1570, 2004. doi:10.1109/TKDE.2004.93.

**27** Uzi Vishkin. Prefix sums and an application thereof, April 1 2003. US Patent 6,542,918.

**28** Limin Xiang and Kazuo Ushijima. ANSV problem on bsrs. *Inf. Process. Lett.*, 65(3):135–138, 1998. doi:10.1016/S0020-0190(97)00214-7.

# Improved Algorithms for Online Rent Minimization Problem Under Unit-Size Jobs

**Enze Sun**
The University of Hong Kong, Hong Kong, China

**Zonghan Yang**
Shanghai Jiao Tong University, China

**Yuhao Zhang**
Shanghai Jiao Tong University, China

## Abstract

We consider the Online Rent Minimization problem, where online jobs with release times, deadlines, and processing times must be scheduled on machines that can be rented for a fixed length period of $T$. The objective is to minimize the number of machine rents. This problem generalizes the Online Machine Minimization problem where machines can be rented for an infinite period, and both problems have an asymptotically optimal competitive ratio of $O(\log(p_{\max}/p_{\min}))$ for general processing times, where $p_{\max}$ and $p_{\min}$ are the maximum and minimum processing times respectively. However, for small values of $p_{\max}/p_{\min}$, a better competitive ratio can be achieved by assuming unit-size jobs. Under this assumption, Devanur et al. (2014) gave an optimal $e$-competitive algorithm for Online Machine Minimization, and Chen and Zhang (2022) gave a $(3e+7) \approx 15.16$-competitive algorithm for Online Rent Minimization. In this paper, we significantly improve the competitive ratio of the Online Rent Minimization problem under unit size to 6, by using a clean oracle-based online algorithm framework.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** Online Algorithm, Scheduling, Machine Minimization, Rent Minimization

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.97

**Related Version** *Full Version*: https://arxiv.org/abs/2306.17241

**Funding** *Yuhao Zhang*: supported by National Natural Science Foundation of China, Grant No. 62102251.

**Acknowledgements** The authors would like to thank Minming Li, Pinyan Lu, and Biaoshuai Tao for many insightful discussions on the research topic of this paper.

## 1 Introduction

*Machine Minimization* is a classical scheduling problem in combinatorial optimization. We are given $n$ jobs with release time and deadline to schedule. Each job $j$ has a length $p_j$ and must be assigned to a machine for $p_j$ units of time between its release time $r_j$ and its deadline $d_j$. However, in many practical scenarios, such as cloud computing, we may not need to buy the machines but only rent them for a fixed period of time. This motivates the *Rent Minimization* problem, introduced by Saha [12]. In this problem, we are given a constant $T$, which represents the duration of a machine rent. The goal is to minimize the number of rents we make to process all jobs within their deadlines.

Another related formulation, inspired by nuclear weapon testing, is the *Calibration* problem, proposed by Bender et al. [3]. In this problem, we are given $m$ machines and a set of jobs that must be completed feasibly. However, before using a machine, we need to *calibrate* it. Each calibration, similar to a rent, activates the machine for a time period of $T$. The goal is to minimize the number of calibrations to process all jobs on time. The main


difference between the Calibration and the Rent Minimization problems is that the former restricts us to have at most $m$ machines working in parallel at any given time, while the latter does not have such a constraint. Therefore, the Rent Minimization problem can be regarded as a special case of the Calibration problem when $m = \infty$.

On the other hand, in the cloud rental scenario and many other practical applications, the computing requests usually increase over time and can be modeled as online released jobs. Therefore, we investigate the problem in an online setting. We do not have any prior knowledge about the jobs before their release time, and need to schedule jobs and rent machines online and irrevocably over time. The goal is to minimize the total number of *rents* for scheduling all jobs. Note that the online generalization is also studied in the Calibration problem by Chen and Zhang [5]. To ensure that online algorithms can schedule all jobs, they also assume $m = \infty$ in their model, which coincides with the Online Rent Minimization model.

**Why consider unit-size jobs?** Saha [12] proposes an $O(\log(p_{\max}/p_{\min}))$-competitive algorithm for the Online Machine Minimization problem. By paying a constant factor, it can be extended to an $O(\log(p_{\max}/p_{\min}))$-competitive algorithm for the Online Rent Minimization problem. ($p_{\max}$ and $p_{\min}$ are the longest and shortest processing time among all jobs.), which was proved to be the best competitive ratio asymptotically. However, in many real-world applications, one company usually receives similar length requests, so the ratio between $p_{\max}$ and $p_{\min}$ may not be too large; and it is worthwhile to reduce the constant factor of the ratio when $p_{\max}/p_{\min}$ is small. To this end, we focus on the special case of unit-size jobs (i.e., all $p_j = 1$). Note that by partitioning jobs by their length into $\log(p_{\max}/p_{\min})$ groups, the $\alpha$-competitive unit-size algorithm can be extended to a roughly $(3\alpha \log(p_{\max}/p_{\min}))$-competitive algorithm in the general case.

The unit-size special case has also been considered in the Online Machine Minimization problem [5, 8, 10]. Devanur et al. [8] present an $e$-competitive algorithm for the Online Machine Minimization problem under unit-size jobs (though earlier work by Bansal et al. [2] implies the same result), and it is the optimal ratio among all deterministic algorithms. Current best lower bound of the online renting problem under unit-size jobs is also $e$ since it is a generalized model. On the positive side, Chen and Zhang [5] study the online renting problem under unit-size jobs. They improve the implicit constant ratio by Saha [12] to $3e + 7 \approx 15.16$. In their algorithm, jobs are distinguished by whether they are long or short based on the length of their time window (i.e., $d_j - r_j$) and are handled separately. Our paper significantly improves the competitive ratio to 6 with a cleaner oracle-based algorithm without identifying whether jobs are short or long.

▶ **Theorem 1.** *There exists an efficient* 6*-competitive algorithm for the online renting problems under unit-size jobs.*

**Our techniques.** In the work of Chen and Zhang [5], they rent machines for long and short jobs separately; as a result, their final competitive ratio is the sum of two cases, which makes the ratio large. The technical reason behind this result is that they use the $e$-competitive Online Machine Minimization algorithm by Devanur et al. [8] as a black box, which is only suitable for *short* jobs. (Roughly speaking, it is because we can view $T = \infty$ when jobs are short.) Therefore, they must use another approach to handle long jobs.

In our paper, we formalize and extend the Online Machine Minimization algorithm to an oracle-based framework, instead of using the algorithm as a black box. The oracle-based framework uses an offline algorithm to guide our online decision. Note that the Online

Machine Minimization algorithm also uses an efficient offline optimal algorithm as an oracle. However, we do not know a polynomial offline Rent Minimization algorithm for unit-size jobs. The main algorithmic novelty is that we find an efficient substitute for the optimal algorithm to act as a bridge between online decisions and the optimal offline solution. The oracle is a kind of optimal augmentation algorithm. It is allowed to use a rent length of $3T$, and the rent number is at most $\mathsf{OPT}$ with rent length $T$. It also satisfies some online monotone properties so that we can control the cost of the online algorithm. Finally, we prove that by following the offline oracle and paying a factor of 6 online, we can recover the same ability for scheduling jobs as the offline oracle. This concludes the competitive ratio of 6.

**Extension to the model with delay.** Chen and Zhang [5] also raise a perspective that the operation *rent* (a.k.a. *calibration* in their paper) needs a non-negative time $\lambda$ to finish. We call it *Online Rent Minimization with Delay*. They propose an $(3(e+1)\lambda+3e+7) \approx (11.15\lambda+15.16)$-competitive algorithm. We use a black box reduction to extend the algorithm in Theorem 1 and improve the ratio to $6(\lambda+1)$.

▶ **Theorem 2.** *As a corollary of Theorem 1, there exists an efficient $6(\lambda+1)$-competitive algorithm when we need $\lambda$ time to finish each rent.*

**Other related works.** Offline Machine Minimization is a well-studied and classic model. Garey and Johnson [9] shows that it is NP-hard. On the algorithm side, Raghavan and Thompson [11] propose an $O(\frac{\log n}{\log\log n})$-approximation algorithm. Later, the ratio has been improved to $O(\sqrt{\frac{\log n}{\log\log n}})$ by Chuzhoy et al. [6]. Whether there exists a constant approximation ratio is still open. Moreover, several special cases are also discussed. Cieliebak et al. [7] focus on the case that each job's active time $(d_j - r_j)$ is small. Yu and Zhang [13] achieve a ratio of 2 in the equal release time case and a ratio of 6 in the equal processing time case.

Scheduling to minimize the number of calibrations is proposed by Bender et al. [3]. The general case is NP-hard even for checking feasibility. Under unit-size jobs, Bender et al. give a 2-approximation algorithm; later, Chen et al. [4] give the first PTAS algorithm. However, it is worth noting that whether the unit-size special case is polynomially solvable is still open. Moreover, Angel et al. [1] introduce the concept of *delay*, which means that each calibration requires $\lambda$ time to finish. They study the delay setting on the one-machine special case of the offline calibration problem and show that it is polynomially solvable.

## 2 Preliminaries

We first define the models and introduce the basic notations.

**Rent Minimization.** We have a set of jobs $\mathcal{J} = \{1, \cdots, n\}$ and a fixed rent length $T$. For job $j \in \mathcal{J}$, it has a release time $r_j$, a deadline $d_j$, and a unit processing time $p_j = 1$. Each job should be assigned to one active machine at an integer time unit $[t, t+1)$, where $t$ is an integer such that $r_j \le t \le d_j - 1$. We can rent a machine at any integer time point $t$. Then we will have an active machine during $[t, t+T)$. The objective is to minimize the number of machine rents to process all jobs in $\mathcal{J}$.

**Online Rent Minimization.** In the online version, all jobs are released online, and they become *visible* at their release time. On the other hand, we need to make rent decisions and assign jobs online irrevocably. In particular, at an integer time point $t$, we have:

- Jobs with release time equal to $t$ become visible.
- We can decide to rent a machine at $t$ or any time after that.
- We can schedule jobs on active machines during the time unit $[t, t+1)$ irrevocably.

**Notations on Rent Set.** We use a multiset $I = \{[s_1, c_1), [s_2, c_2), \cdots [s_i, c_i), \cdots\}$ to denote a set of rents, where the $i$-th rent interval starts at $s_i$ and is active in $[s_i, c_i)$. Note that $c_i$ always equals $s_i + T$ when the rent length is $T$; however, we use the general notation for future reference.

Focusing on the time unit $[t, t+1)$, the number of *active units* $A_I(t)$ is defined as the number of active machines at time $t$, which means that we can schedule at most $A_I(t)$ jobs at time $t$. For a given rent set $I$, we have $A_I(t) = |\{[s_i, c_i) \in I \mid t \in [s_i, c_i)\}|$. We also extend the notation for intervals, such that $A_I(r^*, d^*) = \sum_{t=r^*}^{d^*-1} A_I(t)$ is the number of active units during the time interval $[r^*, d^*)$.

**Feasibility of Rent Set.** We call a rent set $I$ *feasible* for $J$, if we can schedule all jobs in $J$ on $I$. We introduce a lemma based on Hall's Theorem to check whether $I$ is feasible.

For a given jobs set $J$, we define $J(r^*, d^*) = \{j \in J | r^* \le r_j < d_j \le d^*\}$ to represent the jobs that must be assigned inside the interval $[r^*, d^*)$.

▶ **Lemma 3** (Feasibility). *$I$ is feasible for $J$ iff. $\forall r^* \le d^* \in \mathbb{N},\ A_I(r^*, d^*) \ge |J(r^*, d^*)|$ .*

**Proof.** For any fixed $r^*$ and $d^*$, the sum of active units provided by $I$ is $A_I(r^*, d^*)$. Each job released and due between this period must be scheduled on these time slots. If there exists a pair of $r^*$ and $d^*$ such that $A_I(r^*, d^*) < |J(r^*, d^*)|$, there is no feasible assignment because of the pigeonhole principle. On the other hand, if the inequality holds for all $r^*$ and $d^*$, we can view it as a bipartite matching between jobs and active units. There is a feasible assignment by Hall's Theorem. ◀

**An Efficient Checker and Scheduler: Earliest Deadline First (EDF).** **E**arliest **D**eadline **F**irst is a greedy algorithm that can find a feasible assignment for $J$ on $I$ if and only if $I$ is feasible for $J$. When we call $\mathsf{EDF}(J, I)$, we scan time units from early to late, and assign the released job with the earliest deadline to a free active machine at the current time unit. If a job with deadline $d$ cannot find a free active machine at the time unit $[d-1, d)$, $\mathsf{EDF}(J, I)$ fails, and we call $d$ the *fail time* of $\mathsf{EDF}(J, I)$. Otherwise, $\mathsf{EDF}(J, I)$ succeeds. Bender et al. [3] has already proved that EDF can check the feasibility. It is also worth noting that EDF can be efficiently implemented in $O(n \log n)$ by using a heap, instead of going through all integer time points directly.

▶ **Lemma 4** ([3]). *$\mathsf{EDF}(J, I)$ succeeds, i.e., it can find a feasible schedule for $J$ on $I$, if and only if $I$ is feasible for $J$.*

**Using EDF online.** Note that we only make comparisons between released jobs. Therefore, the EDF algorithm can be simulated online : we only need to find a feasible rent set $I$, and then EDF can automatically find a feasible assignment online.

## 3 Oracle-based Online Algorithm Framework

Moving towards online algorithms, one natural way is to use an offline algorithm as an *oracle* to suggest the actions of online algorithms. We keep track of this offline algorithm and make corresponding online decisions when the offline algorithm changes along with the online jobs

release. Whenever the offline algorithm increases by one at the moment $t$ because of the change of the job set, the online algorithm performs one Batch-Rent at this time $t$, which is a fixed rent scheme that contains $\Gamma$ machines. Intuitively, we use these $\Gamma$ machines to catch up with the one increment of the offline oracle. It is worth noting that the $e$-competitive algorithm for Online Machine Minimization follows this approach [8].

In our case, compared to Machine Minimization, we have two main differences in our oracle-based framework. The first difference is the job set we input to the oracle. Because of the online fashion, the most natural way is to input the set of all released jobs to the oracle. In Machine Minimization, it works because $T = \infty$ and earlier rents are always more powerful; however, this is not true in Rent Minimization. Indeed, too early rents may cause trouble in Rent Minimization. Intuitively, we are only allowed to make online rent when the offline oracle reports an increment if we want to bound the competitive ratio in the framework. Consider the case where $T = 10$ and two jobs will be due at 100 with release time 0 and 90. If we report the first job to the offline oracle at 0, the offline oracle will return one new rent interval. Following the oracle-based framework, we will make $\Gamma$ online rent intervals at 0. However, we still need to make more rent intervals at 90, while the offline oracle may not increase. To this end, we use the job set $J_t$ as the job set we input to the oracle at time $t$. A job $j$ is in $J_t$ if it satisfies the following two properties:

**1)** It is *visible* at $t$, i.e., $r_j \le t$.
**2)** It is *emergent* at (or before) $t$, i.e., $d_j \le t + T$.

Another difference is an augmentation factor $\alpha$. The oracle is allowed to have $\alpha T$ active time for each rent. Since the existence of a polynomial time optimal offline algorithm for Rent Minimization is still unknown, this factor allows us to find an efficient substitute. We use $\mathsf{Oracle}_\alpha(J, T)$ (instead of $\mathsf{Oracle}(J, \alpha T)$, since the target rent length is still $T$) to denote an oracle with augmentation factor $\alpha$. The framework is formalized in Algorithm 1.

**Algorithm 1** Oracle-based Online Algorithm Framework.

**procedure** OracleBasedOnline($t$: time, $J$: known jobs, $T$: length of rent)
  $\Delta_t = |\mathsf{Oracle}_\alpha(J_t, T)| - |\mathsf{Oracle}_\alpha(J_{t-1}, T)|$ ▷ $\alpha$ is a positive integer
  Perform $\Delta_t$(if $\Delta_t > 0$) Batch-Rent operations at $t$, consisting of $\Gamma$ machines that start at or after $t$.
  **schedule** jobs at $t$ following $\mathsf{EDF}(J, I)$, where $I$ is the current online rent set.
**end procedure**

Then, we discuss how this framework helps us control the number of rents made by the online algorithm. First, as a substitute for the optimal offline algorithm, $\mathsf{Oracle}_\alpha$ needs to maintain some properties similar to those of the optimal offline algorithm. Second, Batch-Rent should support the online algorithm to be as powerful as the offline oracle in scheduling all released jobs. We integrate and formalize these messages in the following lemma.

▶ **Lemma 5.** *Let* $\mathsf{OPT}(J, T)$ *be the number of rents used by the optimal offline algorithm to schedule the job set* $J$*, Algorithm 1 is* $\Gamma$*-competitive if these three properties are guaranteed:*

**1)** *For any job set* $J$*,* $|\mathsf{Oracle}_\alpha(J, T)| \le \mathsf{OPT}(J, T)$*;*
**2)** *The offline oracle is online monotone:* $|\mathsf{Oracle}_\alpha(J_{t_1}, T)| \le |\mathsf{Oracle}_\alpha(J_{t_2}, T)|$ *if* $t_1 \le t_2$*;*
**3)** *Algorithm 1 is feasible for scheduling all online released jobs.*

**Proof.** The online algorithm makes $\sum_{\Delta_t > 0} \Delta_t$ rent batches, which is exactly $|\mathsf{Oracle}_\alpha(\mathcal{J}, T)|$ by property 2) and is not greater than $|\mathsf{OPT}(\mathcal{J}, T)|$ by property 1). Also, the output satisfies the feasibility requirement by property 3). Therefore, Algorithm 1 is $\Gamma$-competitive. ◀

**The $e$-competitive algorithm for Online Machine Minimization.** We can use the framework to understand the $e$-competitive Online Machine Minimization algorithm.

- Oracle is the optimal offline algorithm, and we set $\alpha = 1$. The monotonicity directly comes from optimality.
- $J_t$ is the set of visible jobs at $t$ because all jobs are emergent.
- Each Batch-Rent contains $e$ new machines in average; for simplicity, we omit any rounding issues related to $e$.

**Choice of the oracle.** Recall that we do not have an efficient optimal algorithm for Rent Minimization currently. It remains to find a suitable substitute that also uses a small number of rents (property 1). One candidate algorithm may be the Lazy-Binning algorithm by Bender et al. [3], which requires an augmentation factor of 2 to satisfy property 1). However, Lazy-Binning algorithm, as well as other relatively simple 2 approximation algorithms we come up with, cannot guarantee monotonicity. This will make us fail to bound the competitive ratio of the online algorithm. In the next section, we introduce our oracle with an augmentation factor of 3, called the semi-online algorithm, which provides all the properties we need in Lemma 5.

## 4 The Semi-Online Algorithm

In this section, we introduce the semi-online algorithm shown as Algorithm 2, which uses an augmentation factor of 3 and acts as the $\mathsf{Oracle}_3$ in our framework.

**Algorithm 2** The Semi-Online Algorithm.

```
procedure SemiOnline(J: input job set, T: length of rent)
    J', I ← ∅                                   ▷ I is a multiset for rents.
    τ_j = max{r_j, d_j − T} for all j.
    for j ∈ J in non-decreasing order of τ_j do
        J' ← J' ∪ {j}
        if EDF(J', I) fails then
            I ← I ∪ {[τ_j − T, τ_j + 2T)}       ▷ A rent that starts at τ_j − T with length 3T.
        end if
    end for
    return I
end procedure
```

We call Algorithm 2 semi-online, because the enumerating order is exactly the same as how $J_t$ increases in the oracle-based framework. Thus, if we have some new jobs with $r_j = t$ or $d_j - T = t$ when the online time moves from $t-1$ to $t$, the only possible difference between $\mathsf{SemiOnline}(J_{t-1}, T)$ and $\mathsf{SemiOnline}(J_t, T)$ is some rent intervals of $[t-T, t+2T)$. This observation could be formalized into the following properties of the semi-online algorithm.

▶ **Lemma 6** (Strong Monotonicity). *We have the following two properties for Algorithm 2.*

1. *For any $J_{t_1}$ and $J_{t_2}$ where $t_1 \le t_2$, we have $\mathsf{SemiOnline}(J_{t_1}, T) \subseteq \mathsf{SemiOnline}(J_{t_2}, T)$.*
2. *$\mathsf{SemiOnline}(J_t, T) \setminus \mathsf{SemiOnline}(J_{t-1}, T)$ is a multiset of a fixed rent interval $[t-T, t+2T)$.*

**Proof.** Intuitively, the reason behind the lemma is that the order of $\tau_j$ is the same as the order in which we insert jobs into $J_t$ as $t$ increases. Formally speaking, compare $J_{t_1}$ and $J_{t_2}$ and consider a job $j \in J_{t_2} \setminus J_{t_1}$. By definition, we have that $\tau_j \ge \max_{j' \in J_{t_1}} \tau_{j'}$. Therefore,

Algorithm 2 first enumerates the jobs in $J_{t_1}$ and then the jobs in $J_{t_2} \setminus J_{t_1}$, which concludes the first property immediately. For the second property, the reason is that $J_t \setminus J_{t-1}$ is a set of jobs with $r_j = t$ or $d_j - T = t$. In other words, we enumerate them after jobs in $J_{t-1}$. Thus, if $I$ continues to grow when we enumerate them, the new interval must be $[t-T, t+2T)$. ◀

The strong monotonicity in Lemma 6 suffices to show the weak monotonicity in property 2) of Lemma 5. On the other hand, these two properties are also used in the proof of property 3) later. It remains to prove property 1) by bounding the cardinality of the semi-online algorithm's solution.

▶ **Lemma 7.** $|\mathsf{SemiOnline}(J,T)| \le \mathsf{OPT}(J,T)$, *and* $\mathsf{SemiOnline}(J,T)$ *is feasible for* $J$.

Before proving Lemma 7, we introduce coOPT so that we can better understand the solution structure.

▶ **Definition 8** (Optimal complement solution). *For a job set $J$, rent length $T$, and a given rent set $I$ (which may not be length $T$), the optimal complement solution of $I$, denoted as* $\mathsf{coOPT}(I)$, *is defined as a rent set of length $T$ with minimum cardinality such that $I \cup \mathsf{coOPT}(I)$ is feasible for $J$.*

▶ **Fact 9.** $\mathsf{coOPT}(\varnothing) = \mathsf{OPT}, \mathsf{coOPT}(\mathsf{OPT}) = \varnothing$.

Consider a rent set $I$ that is infeasible for $J$. Below, we state the main property of coOPT.

▶ **Lemma 10.** *Let $d$ be the fail time of* $\mathsf{EDF}(J,I)$. *There exists a* $\mathsf{coOPT}(I)$ *such that there is a rent interval $[s, s+T) \in \mathsf{coOPT}(I)$ that satisfies: $d - T \le s < d$.*

**Proof.** We use coOPT as a shorthand for $\mathsf{coOPT}(I)$ in this proof. Let $[s, s+T)$ be the earliest interval in coOPT.

First, we show that $s < d$. Suppose, for a contradiction, that $s \ge d$. Let $j$ be the job that fails in $\mathsf{EDF}(J,I)$, where $J$ is a fixed given job set. Then, $j$ has no more active units in coOPT, since all rent intervals start at or after $d$. But this contradicts the definition of coOPT, which is a feasible rent set for $J$.

Second, we show that $s \ge d - T$ can be true. If this is not true, we construct a new rent set $\mathsf{coOPT}' = \mathsf{coOPT} \setminus \{[s, s+T)\} \cup \{[d-T, d)\}$. That is, we replace the rent interval at $s$ with another one at $d - T$. We claim that $\mathsf{coOPT}' \cup I$ is also feasible for $J$. This means that coOPT′ is also a feasible coOPT, and $[s' = d - T, T)$ is a feasible rent interval that satisfies our desired condition.

To prove our claim, we fix an arbitrary choice of $r^* \le d^* \in \mathbb{N}$, and we verify the condition in Lemma 3, i.e., $A_{I \cup \mathsf{coOPT}'}(r^*, d^*) \ge |J(r^*, d^*)|$.

We consider two cases:

- Case 1: $d^* < d$. If the condition does not hold, we have $A_I(r^*, d^*) \le A_{I \cup \mathsf{coOPT}'}(r^*, d^*) < J'(r^*, d^*)$, which means that $\mathsf{EDF}(I, J')$ must fail no later than $d^*$ since the active units are not enough beforehand. This contradicts the definition of $d$.
- Case 2: $d^* \ge d$. The only difference between coOPT and coOPT′ is the contribution of active units by $[s, s+T)$ and $[d-T, d)$. We prove that $[d-T, d)$ must provide at least as many active units as $[s, s+T)$ does. Referring to Figure 1, we see that $[d-T, d)$ has more active units than $[s, s+T)$ in $[s+T, d)$, and vice versa in $[s, d-T)$. Since $d^* \ge d$, we need $r^* \le d - T$ to reach the advantage area of coOPT; however, $[r^*, d^*)$ then covers the whole part of $[d-T, d)$. This implies that the total contribution of coOPT never exceeds that of coOPT′.

The discussion concludes the claim. ◀

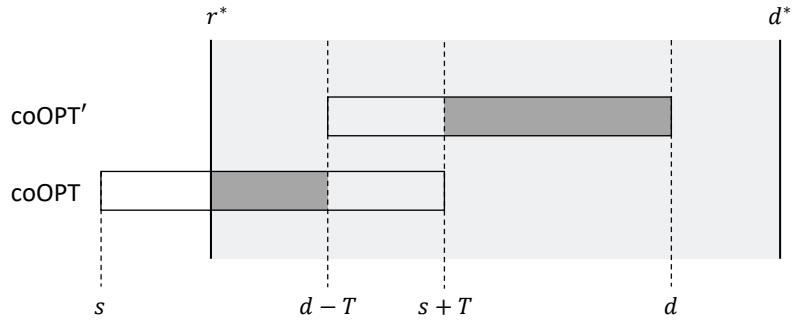



**Figure 1** coOPT' contributes at least as many active units as coOPT on $[r^*, d^*)$ when $d^* \geq d$.

▶ **Lemma 11.** *At the end of each iteration of $j$, $I$ is feasible for $J'$.*

**Proof.** We prove it by induction. In the base case, $I$ is feasible for $J'$ when they are $\varnothing$. Then, assume the lemma is true after the $(j-1)$-th iteration. At the $j$-th iteration, we add a job $j$ to $J'$. This means that $\forall r^* \leq r_j,\ d^* \geq d_j$, $|J'(r^*, d^*)|$ will increase by one. If $\mathsf{EDF}(J', I)$ is already feasible, we are done. Otherwise, the algorithm will employ a new $3T$ length rent interval $[\tau_j - T, \tau_j + 2T)$. Notice that $d^* \geq d_j \geq \tau_j$. Every $A_I(r^*, d^*)$ also increases by at least one. Thus, we still have $A_I(r^*, d^*) \geq |J'(r^*, d^*)|$ after we employ $[\tau_j - T, \tau_j + 2T)$ (at the end of the $j$-th iteration). ◀

▶ **Corollary 12.** $\mathsf{SemiOnline}(J, T)$ *is feasible for $J$.*

▶ **Lemma 13.** *In the $j$-th iteration, if $\mathsf{EDF}(J', I)$ is infeasible in Algorithm 2, before we rent $[t - T, t + 2T)$, we have*

$$|\mathsf{coOPT}(I \cup \{[t - T, t + 2T)\})| \leq |\mathsf{coOPT}(I)| - 1.$$

**Proof.** By the condition of the lemma and Lemma 11, we have $\mathsf{EDF}(J', I)$ is infeasible while $\mathsf{EDF}(J' \setminus \{j\}, I)$ is feasible. By the enumerating order, all the jobs $j'$ in $J'$ must satisfy $\max\{r_j, d_{j'} - T\} \leq \tau_j$. Therefore, $\mathsf{EDF}(J', I)$ must fail at a deadline $d \leq \tau_j + T$. On the other hand, for all $j' \in J$ such that $d_{j'} < \tau_j$, we must have $j' \in J' \setminus \{j\}$, also because of the enumerating order. Thus, we can show that $d \geq \tau_j$. Otherwise, $J' \setminus \{j\}$ would be infeasible for $I$, which is a contradiction. In conclusion, we show that the failure time $d$ of $\mathsf{EDF}(J', I)$ satisfies $\tau_j \leq d < \tau_j + T$.

Finally, by Lemma 10, there exists a coOPT with rent interval $[s, s + T]$ such that $d - T \leq s < d$. It implies that $\tau_j - T < s < \tau_j + T$. Therefore $[s, s + T)$ is always a subset of $[\tau_j - T, \tau_j + 2T)$. We have

$$|\mathsf{coOPT}(I \cup \{[t - T, t + 2T)\})| \leq |\mathsf{coOPT}(I \cup \{s\})| = |\mathsf{coOPT}(I)| - 1.$$ ◀

**Proof of Lemma 7.** Recall Fact 9 that $\mathsf{coOPT}(\varnothing) = \mathsf{OPT}$. It follows that SemiOnline rents at most OPT times as a corollary of Lemma 13. ◀

## 5 The 6-competitive Online Algorithm

It remains to define the rent scheme for each Batch-Rent in the framework. For completeness, we formally describe the algorithm in Algorithm 3.

**Algorithm 3** The Online algorithm.

**procedure** OnlineRent($t$: time, $J$: known jobs, $T$: length of rent)
  $\Delta = |\mathsf{SemiOnline}(J_t, T)| - |\mathsf{SemiOnline}(J_{t-1}, T)|$
  Perform $\Delta$ Batch-Rent at time $t$, each consists of 6 machines: 4 at $t$ and 2 at $t+T$.
  **schedule** jobs at $t$ following $\mathsf{EDF}(J, I)$, where $I$ is the current online rent set.
**end procedure**

Next, we prove the property 3) of Lemma 5 by our design of Batch-Rent, i.e., to show Algorithm 3 is feasible for the total job set $\mathcal{J}$. Combining with the property 1) and 2) by the semi-online algorithm, we can conclude our online algorithm is 6-compeititve as claimed in Theorem 1.

First, we introduce an obvious relationship between online and semi-online algorithms.

▶ **Fact 14.** *At every moment $t$, there always exists a bijection from one semi-online rent batch to one online rent batch, such that both batches are at the same time:* $4 \times [t, t+T) + 2 \times [t+T, t+2T)$ *in* Online $\mapsto [t-T, t+2T)$ *in* SemiOnline.

**Proof.** This fact is directly implied by the second property of Lemma 6. Whenever $\mathsf{SemiOnline}(J_t, T)$ increases from $\mathsf{SemiOnline}(J_{t-1}, T)$ by some rent intervals of $[t-T, t+2T)$, the new batches must be $4 \times [t, t+T) + 2 \times [t+T, t+2T)$. Each of them can correspond to one $[t-T, t+2T)$. ◀

We prove the feasibility by showing that the active units provided by the online algorithm are always enough for the possible jobs inside any possible interval $[r^*, d^*)$.

▶ **Lemma 15.** *Let $I$ be the rent sets made by our algorithm. We have $\forall r^* \le d^* \in \mathbb{N}$, $A(r^*, d^*) \ge |\mathcal{J}(r^*, d^*)|$, where we use $A$ to denote $A_I$ for simplicity.*

We remark that Lemma 15 provides the necessary information to prove the correctness via the feasibility lemma (Lemma 3). It only remains to complete the proof of Lemma 15.

## 5.1 Proof of Lemma 15

Let us fix an arbitrary range $r^* \le d^*$, and discuss $A(r^*, d^*)$ and $\mathcal{J}(r^*, d^*)$ separately. First, we discuss $A(r^*, d^*)$. The behavior of the online algorithm can be represented by a set of online rent batches. Note that only those rent batches that start in $(r^* - 2T, d^*)$ can provide active units inside $[r^*, d^*)$. Therefore, we only discuss a subset $B$ of all rent batches with a start time in $(r^* - 2T, d^*)$. Each $b$ means a Batch-Rent made by Algorithm 3. We use $t(b)$ to mean its decision time. That is, a batch $b$ contains 4 rent intervals of $[t(b), t(b)+T)$, and 2 of $[t(b)+T, t(b)+2T)$.

We partition the time interval $[r^*, d^*)$ by several critical time points. The first time point is $\theta_1 = \max\left\{\left(\frac{r^*+d^*}{2}\right), d^* - T\right\}$. It means that $\theta_1 = d^* - T$ when $d^* - r^* \ge 2T$ and $\theta_1 = \left(\frac{r^*+d^*}{2}\right)$ when $d^* - r^* < 2T$. Remark that in both cases, $\theta_1 \ge d^* - T$. Then we recursively define $\theta_{i+1} = \left(\frac{\theta_i + d^*}{2}\right)$ for every $i \ge 1$ until $\lfloor \theta_i \rfloor = d^* - 1$. Besides, we let $\theta_0 = r^* - 1$. For $i \ge 1$, we call $[\lfloor \theta_{i-1} \rfloor + 1, \lfloor \theta_i \rfloor]$ the $i$-th sub-interval, which is the minimal sub-interval of $(\theta_{i-1}, \theta_i]$ that contains all integers in it. We define $B_i \subseteq B$ as the set of rent batches starting in the $i$-th sub-interval. Moreover, we let $B_0$ be the set of rent batches starting in $(r^* - 2T, r^*]$.

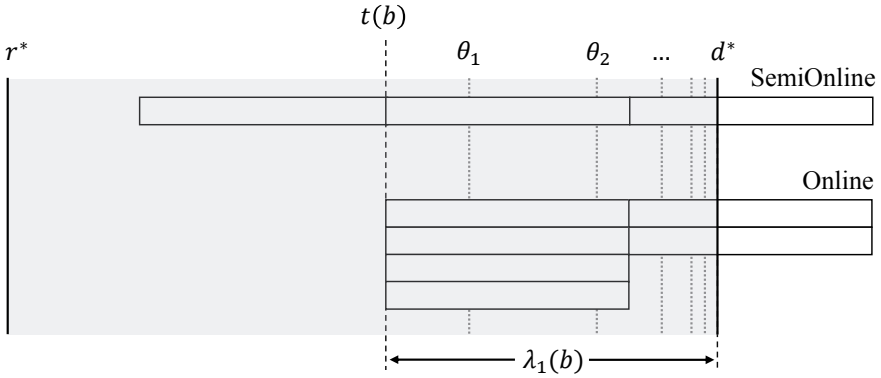



**Figure 2** An example for $b \in B_1$ when $d^* - r^* > 2T$. The shaded area is the considered $[r^*, d^*)$, and the online rent batch provides $2\lambda_1(b) + 2\min\{\lambda_1(b), T\}$ active units to it.

For each $b \in B$, recall that $t(b)$ is the time it was allocated, and we let $\lambda_i(b)$ be the length of the intersecting interval of $[t(b), t(b)+2T)$ and $[\lfloor\theta_{i-1}\rfloor + 1, d^*)$. Note that $\lambda_1(b)$ represents the intersecting interval with the whole $[r^*, d^*)$. Let $L = \min\{2T, d^* - r^*\} = 2(d^* - \theta_1)$ denote the maximum possible length in $\lambda_1$, and by our partition method. It follows the property of the length of sub-intervals by our partition.

▶ **Lemma 16** (Partition length property). *For any batch $b \in B_i$ where $i \geq 1$, the intersection of $[t(b), t(b)+2T)$ and $[\lfloor\theta_{i-1}\rfloor + 1, d^*)$ satisfies: $2^{-i} \cdot L \leq \lambda_i(b) \leq 2^{1-i} \cdot L$.*

**Proof.** For $i = 1$, $\lambda_1(b) = \min\{2T, d^* - t(b)\}$. By definition, $r^* \leq t(b) \leq \lfloor\theta_1\rfloor$, hence

$$L/2 = \min\{T, d^* - \theta_1\} \leq \min\{2T, d^* - t(b)\} \leq \min\{2T, d^* - r^*\} = L.$$

For $i \geq 2$, by definition $d^* - \theta_i = 2^{-i} \cdot L$. Because $t(b) \in [\lfloor\theta_{i-1}\rfloor + 1, \lfloor\theta_i\rfloor] \subseteq (\theta_{i-1}, \theta_i]$, we have $2^{-i} \cdot L \leq \lambda_i(b) \leq \min\{2T,\ 2^{1-i} \cdot L\}$. Since $2 \cdot L \leq 2T$, we conclude the lemma. ◀

Then, we present the lemmas for a lower bound of active units and an upper bound of job numbers.

▶ **Lemma 17** (Lower bound of active units).

$$\begin{aligned} A(r^*, d^*) \geq & \sum_{b \in B_0} (2\lambda_1(b) + 2\max\{\lambda_1(b) - T, 0\}) \\ & + \sum_{b \in B_1} (2\lambda_1(b) + 2\min\{\lambda_1(b), T\}) \\ & + \sum_{i \geq 2} 4 \cdot (2^{-i} \cdot L) \cdot |B_i|. \end{aligned}$$

**Proof.** Three terms on the RHS of the inequality are counting of $B_0, B_1$ and $B_{\geq 2}$, where the first two are straightforward counting as shown in Figure 2, and the last term was scaled down a bit by Lemma 16:

$$4 \sum_{i \geq 2} \sum_{b \in B_i} \lambda_1(b) = 4 \sum_{i \geq 2} \sum_{b \in B_i} \lambda_i(b) \geq \sum_{i \geq 2} 4 \cdot (2^{-i} \cdot L) \cdot |B_i|.$$ ◀

Let $\mathcal{J}_i$ be the job set with deadline at most $d^*$ and released in the $i$-th subinterval:

$$\mathcal{J}_i = \{j \in \mathcal{J} \mid \lfloor\theta_{i-1}\rfloor + 1 \le r_j \le \lfloor\theta_i\rfloor,\ d_j \le d^*\}.$$

We provide an upper bound of $\mathcal{J}_i$ by the performance of our algorithm.

▶ **Lemma 18.** *Let $I_t$ be the semi-online batches allocated at or before time $t$. We have that $\mathcal{J}_i$ is no more than the active units after $\lfloor\theta_{i-1}\rfloor + 1$ provided by* SemiOnline *at $\lfloor\theta_i\rfloor$. i.e.,*

$$\mathcal{J}_i \le A_{I_{\lfloor\theta_i\rfloor}}(\lfloor\theta_{i-1}\rfloor + 1, d^*).$$

**Proof.** Let us observe the time point $t = \lfloor\theta_i\rfloor$. SemiOnline$(J_t, T)$ reports $I_t$ at this time. By Lemma 11, $I_t$ is feasible for $J_t$. By the definition of $\theta_i$, we prove $t \ge d^* - T$ because $\theta_1 \ge d^* - T$. Thus, all jobs with deadlines at most $d^*$ are already in $J_t$, and combining with the feasibility of $I_t$, we have:

$$\mathcal{J}_i = J_t(\lfloor\theta_{i-1}\rfloor + 1, d^*) \le A_t(\lfloor\theta_{i-1}\rfloor + 1, d^*).$$

The lemma then concludes because we define $t = \lfloor\theta_i\rfloor$. ◀

▶ **Lemma 19** (Upper bound of job numbers)**.** *We have two different upper bounds for $\mathcal{J}_i$:*

- $i = 1$: $\quad |\mathcal{J}_1| \le \sum_{b\in B_0} \lambda_1(b) + \sum_{b\in B_1} (\lambda_1(b) + L/2)$.
- $i \ge 2$: $\quad |\mathcal{J}_i| \le \sum_{b\in B_0} \lambda_i(b) + 2\cdot(2^{-i}\cdot L)\cdot \sum_{j=1}^{i} |B_j|$.

**Proof.** In this proof, we apply Lemma 18 and count the number of active units after $\theta_{i-1}$ by $I_{\lfloor\theta_i\rfloor}$. Note that by Fact 14, each $3T$ length rent interval in $I_{\lfloor\theta_i\rfloor}$ corresponds to an online Batch-Rent.

First, let us consider the case $i = 1$. $I_{\lfloor\theta_i\rfloor}$ corresponds to the online batches with $t(b) \le I_{\lfloor\theta_i\rfloor}$. Note that the ending time of $b$ and its corresponding semi-online rent interval are both $t(b) + 2T$. Thus, $A_{I_{\lfloor\theta_i\rfloor}}(\lfloor\theta_{i-1}\rfloor + 1, d^*)$ corresponds to $B_0$ and $B_1$.

For $i = 1$, we keep the $B_0$ straightforward and calculate the upper bound of active units provided by the corresponding semi-online batch for each batch $b$ in $B_1$. Note that the semi-online rent set spans $[t(b) - T, t(b) + 2T)$. We split the $3T$ interval into first $T$ and last $2T$: the latter could be upper bounded by $\lambda_1(b)$ using Lemma 16, and we could find that the former is at most $L/2$:

- When $d^* - r^* < 2T$, $t(b) - r^* < (d^* - r^*)/2$, then $\min\{T, t(b) - r^*\} < L/2$.
- When $d^* - r^* \ge 2T$, $L = 2T$, and then $\min\{T, t(b) - r^*\} \le T = L/2$.

So we can conclude that

$$|\mathcal{J}_1| \le \sum_{b\in B_0} \lambda_1(b) + \sum_{b\in B_1} (\lambda_1(b) + L/2).$$

For the case $i \ge 2$, all jobs released in sub-interval $i$ can be allocated at most $[\theta_{i-1}+1, d^*)$, and by Lemma 16 each semi-online batch covers at most $2\cdot(2^{-i}\cdot L)$. Like $i = 1$, the inequality follows direct counting on $\cup_{j=0}^{i} B_j$. ◀

Thus far, we are ready to prove Lemma 15 by a charging argument.

**Proof of Lemma 15.** Recall that in Lemma 17 we have

$$A(r^*, d^*) \geq \sum_{b \in B_0} (2\max\{\lambda_1(b) - T, 0\} + 2\lambda_1(b)) + \sum_{b \in B_1} (2\lambda_1(b) + 2\min\{T, \lambda_1(b)\}) + \sum_{i \geq 2} 4 \cdot (2^{-i} \cdot L) \cdot |B_i|. \tag{1}$$

Also, by Lemma 19,

$$\mathcal{J}(r^*, d^*) \leq \sum_{b \in B_0} \lambda_1(b) + \sum_{b \in B_1} (\lambda_1(b) + L/2) + \sum_{i \geq 2} \left( \sum_{b \in B_0} \lambda_i(b) + 2 \cdot (2^{-i} \cdot L) \cdot \sum_{j=1}^{i} |B_j| \right). \tag{2}$$

Using these two inequalities, we charge the upper bound of $\mathcal{J}(r^*, d^*)$ and the lower bound of $A(r^*, d^*)$ to each $b \in B$. We prove that for each $b$, the charged amount of $\mathcal{J}(r^*, d^*)$'s upper bound is at most $A(r^*, d^*)$'s lower bound.

First, for $b \in B_0$, the contribution of $b$ to the lower bound of $A(r^*, d^*)$ (i.e., RHS of Equation (1)) is: $2\max\{\lambda_1(b) - T, 0\} + 2\lambda_1(b)$. The contribution to the upper bound of $\mathcal{J}(r^*, d^*)$ (i.e., RHS of Equation (2)) is: $\lambda_1(b) + \sum_{i \geq 2} \lambda_i(b)$. Note that $b \in B_0$ only contributes to a prefix of $[r, d)$, it is easy to see that $\lambda_i(b) \leq 2^{1-i}\lambda_1(b)$, and thus

$$\lambda_1(b) + \sum_{i \geq 2} \lambda_i(b) \leq \lambda_1(b) + \sum_{i \geq 2} 2^{1-i}\lambda_1(b) < 2\lambda_1(b).$$

We are done for $b \in B_0$.

Second, for $b \in B_1$, the contribution of $b$ to the lower bound of $A(r^*, d^*)$ (i.e., RHS of Equation (1)) is: $2\lambda_1(b) + 2\min\{T, \lambda_1(b)\}$. The contribution to the upper bound of $\mathcal{J}(r^*, d^*)$ (i.e., RHS of Equation (2)) is: $\lambda_1(b) + L/2 + \sum_{i \geq 2} 2 \cdot (2^{-i} \cdot L)$. Then, we have

$$\begin{aligned} \lambda_1(b) + L/2 + \sum_{i \geq 2} 2 \cdot (2^{-i} \cdot L) &< \lambda_1(b) + L/2 + 2 \cdot \left(2^{-1} \cdot L\right) \\ &\leq \lambda_1(b) + L/2 + 2 \cdot \min\{T, \lambda_1(b)\} \\ &\leq 2\lambda_1(b) + 2\min\{T, \lambda_1(b)\}. \end{aligned}$$

The last inequality holds by Lemma 16. Therefore, we are done for $b \in B_1$.

Finally, for $b \in B_{i \geq 2}$, the contribution of $b$ to the lower bound of $A(r^*, d^*)$ (i.e., RHS of Equation (1)) is: $4 \cdot (2^{-i} \cdot L)$. The contribution to the upper bound of $\mathcal{J}(r^*, d^*)$ (i.e., RHS of Equation (2)) is: $\sum_{i' \geq i} 2 \cdot (2^{-i'} \cdot L)$. We are done because $\sum_{i' \geq i} 2^{-i'} < 2^{1-i}$. Summing up three parts, we have proved that $A(r^*, d^*) \geq \mathcal{J}(r^*, d^*)$. ◀

## 5.2 A Remark on Running Time

We only need to recalculate SemiOnline if the job set gets updated, and the procedure of reconstructing the rent set in SemiOnline can be maintained incrementally. Thus, it is possible to implement the algorithm calling EDF at most $n + \mathsf{OPT} \leq 2n$ times, and hence achieve a worst case guarantee of $O(n^2 \log n)$. Also, note that a job can influence the calculation of SemiOnline for at most $O(T)$ time units, so the SemiOnline can also update in $O(nw \log w)$ if there are at most $O(w)$ jobs within any interval of length $O(T)$. Therefore, the online algorithm is efficient in terms of worst case guarantee and also average online updating.

## 6 Online Rent Minimization with Delay

In the version with delay, we are also given an online released job set $\mathcal{J}$ and a rent length $T$. We aim to minimize the number of rents needed to process all jobs. The notations are the same as in the Online Rent Minimization problem. Moreover, we are given a nonnegative integer $\lambda$ as the delay parameter. It means that if we rent a machine at time $t$, we will have an active machine at $[t+\lambda, t+\lambda+T)$. Note that it is impossible to serve an unknown emergency job with $d_j - r_j \leq \lambda$ online; following Chen and Zhang [5], we require that the active time $d_j - r_j$ is at least $\lambda + 1$.

We use the following reduction lemma and our 6-competitive no-delay algorithm as a black box to prove Theorem 2. Chen and Zhang [5] also mention this approach.

▶ **Lemma 20** (Reduction). *If* $\mathsf{ALG}(\mathcal{J}) \leq \Gamma \cdot \mathsf{OPT}(\mathcal{J})$ *for every job set* $\mathcal{J}$*, we have an algorithm* $\mathsf{ALG}_\lambda$ *that guarantees*

$$\mathsf{ALG}_\lambda(\mathcal{J}, \lambda) \leq \Gamma \cdot (\lambda + 1) \cdot \mathsf{OPT}(\mathcal{J}),$$

*if* $\forall j \in \mathcal{J}$, $d_j - r_j \geq \lambda + 1$.

After proving Lemma 20, Theorem 2 follows directly. See the full version for a complete proof.

▶ **Theorem 2.** *As a corollary of Theorem 1, there exists an efficient* $6(\lambda+1)$*-competitive algorithm when we need* $\lambda$ *time to finish each rent.*

## 7 Conclusion and Future Work

In conclusion, our main contribution is a 6-competitive algorithm for the Online Rent Minimization problem under unit-size jobs, which follows the oracle-based framework.

Since the Online Rent Minimization problem is a generalization of the Online Machine Minimization problem, where we have an optimal $e$-competitive algorithm, one major question is: Is the Rent Minimization problem strictly harder than the Machine Minimization problem?

On the other hand, we are also interested in the power of oracle-based algorithms. Note that the optimal $e$-competitive algorithm for Machine Minimization follows the oracle-based framework. It is interesting to ask: What is the best competitive ratio we can achieve for the Online Rent Minimization problem by using the oracle-based framework? The Semi-Online captures our current understanding of the possible range of optimal solutions, so replacing it with an optimal oracle cannot improve ratio directly by the same argument in the paper. Is it possible to obtain a better ratio with access to an optimal oracle?

## References


1 Eric Angel, Evripidis Bampis, Vincent Chau, and Vassilis Zissimopoulos. On the Complexity of Minimizing the Total Calibration Cost. In Mingyu Xiao and Frances Rosamond, editors, *Frontiers in Algorithmics*, Lecture Notes in Computer Science, pages 1–12, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-59605-1_1.

2 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), March 2007. doi:10.1145/1206035.1206038.

3 Michael A. Bender, David P. Bunde, Vitus J. Leung, Samuel McCauley, and Cynthia A. Phillips. Efficient scheduling to minimize calibrations. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 280–287, Montréal Québec Canada, July 2013. ACM. doi:10.1145/2486159.2486193.

4 Lin Chen, Minming Li, Guohui Lin, and Kai Wang. Approximation of Scheduling with Calibrations on Multiple Machines (Brief Announcement). In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 237–239, Phoenix AZ USA, June 2019. ACM. doi:10.1145/3323165.3323173.

5 Zuzhi Chen and Jialin Zhang. Online scheduling of time-critical tasks to minimize the number of calibrations. *Theoretical Computer Science*, page S0304397522000548, January 2022. doi:10.1016/j.tcs.2022.01.040.

6 J. Chuzhoy, S. Guha, S. Khanna, and J. Naor. Machine Minimization for Scheduling Jobs with Interval Constraints. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 81–90, Rome, Italy, 2004. IEEE. doi:10.1109/FOCS.2004.38.

7 Mark Cieliebak, Thomas Erlebach, Fabian Hennecke, Birgitta Weber, and Peter Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004), 22-27 August 2004, Toulouse, France*, volume 155 of *IFIP*, pages 209–222. Kluwer/Springer, 2004. doi:10.1007/1-4020-8141-3_18.

8 Nikhil Devanur, Konstantin Makarychev, Debmalya Panigrahi, and Grigory Yaroslavtsev. Online Algorithms for Machine Minimization. *arXiv:1403.0486 [cs]*, March 2014. arXiv:1403.0486.

9 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

10 Mong-Jen Kao, Jian-Jia Chen, Ignaz Rutter, and Dorothea Wagner. Competitive design and analysis for machine-minimizing job scheduling problem. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation – 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, volume 7676 of *Lecture Notes in Computer Science*, pages 75–84. Springer, 2012. doi:10.1007/978-3-642-35261-4_11.

11 Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Comb.*, 7(4):365–374, 1987. doi:10.1007/BF02579324.

12 Barna Saha. Renting a Cloud. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013)*, volume 24 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 437–448, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.FSTTCS.2013.437.

13 Guosong Yu and Guochuan Zhang. Scheduling with a minimum number of machines. *Oper. Res. Lett.*, 37(2):97–101, 2009. doi:10.1016/j.orl.2009.01.008.

# Simple Deterministic Approximation for Submodular Multiple Knapsack Problem

**Xiaoming Sun**
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

**Jialin Zhang**
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China

**Zhijie Zhang**[1]
Center for Applied Mathematics of Fujian Province, School of Mathematics and Statistics, Fuzhou University, China

## Abstract

Submodular maximization has been a central topic in theoretical computer science and combinatorial optimization over the last decades. Plenty of well-performed approximation algorithms have been designed for the problem over a variety of constraints. In this paper, we consider the submodular multiple knapsack problem (SMKP). In SMKP, the profits of each subset of elements are specified by a monotone submodular function. The goal is to find a feasible packing of elements over multiple bins (knapsacks) to maximize the profit. Recently, Fairstein et al. [ESA20] proposed a nearly optimal $(1 - e^{-1} - \epsilon)$-approximation algorithm for SMKP. Their algorithm is obtained by combining configuration LP, a grouping technique for bin packing, and the continuous greedy algorithm for submodular maximization. As a result, the algorithm is somewhat sophisticated and inherently randomized. In this paper, we present an arguably simple deterministic combinatorial algorithm for SMKP, which achieves a $(1 - e^{-1} - \epsilon)$-approximation ratio. Our algorithm is based on very different ideas compared with Fairstein et al. [ESA20].

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

**Keywords and phrases** Submodular maximization, knapsack problem, deterministic algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.98

**Related Version** *Full Version*: https://arxiv.org/abs/2003.11450

**Funding** This work was supported in part by the National Natural Science Foundation of China Grants No. 61832003, 62272441.

## 1 Introduction

The *multiple knapsack problem* (MKP) is defined as follows. We are given a set $N$ of $n$ elements and a set $M$ of $m$ bins (knapsacks). Each element $u \in N$ has a positive cost $c(u) > 0$ and a positive profit $p(u) > 0$. The cost (profit) of a subset $S \subseteq N$ equals the sum of the costs (profits) of its elements. The $j$-th bin in $M$ has a positive budget $B_j > 0$ for $1 \leq j \leq m$. A subset $S \subseteq N$ is feasible if there is a disjoint partition $\{S_j\}_{j=1}^m$ of $S$ such that $c(S_j) \leq B_j$ for $1 \leq j \leq m$. The goal is to find a feasible set $S$ (and its partition $\{S_j\}_{j=1}^m$) whose profit is maximized. It is well-known that the problem admits a PTAS but no FPTAS assuming P $\neq$ NP [18, 5, 17].

In this paper, we consider the submodular generalization of the above problem, referred to as the *submodular multiple knapsack problem* (SMKP). In SMKP, the profit is in general non-additive and specified by a *monotone submodular* function $f : 2^N \to \mathbb{R}_+$. Here, a set function $f : 2^N \to \mathbb{R}$ is monotone if $f(S) \leq f(T)$ for any $S \subseteq T$ and submodular if

[1] Corresponding author

$f(S \cup \{u\}) - f(S) \geq f(T \cup \{u\}) - f(T)$ for any $S \subseteq T$ and $u \notin T$. The goal is again to find a feasible set $S$ which maximizes the profit $f(S)$. When $m = 1$, the problem reduces to submodular maximization under a knapsack constraint, which enjoys an optimal $(1 - e^{-1})$-approximation [18, 25].

Submodular functions capture the effect of diminishing returns in the economy and generalize many well-known functions such as coverage functions, cut functions, matroid rank functions, and log determinants. By introducing a submodular objective, SMKP falls in the field of submodular maximization, which studies maximization problems with submodular objectives, including maximum coverage problem, maximum cut problem, submodular welfare problem [26], influence maximization [19]. The study of submodular maximization has lasted for more than forty years. As early as 1978, it was shown that for monotone submodular maximization, a greedy algorithm achieves a $(1 - e^{-1})$-approximation under the cardinality constraint [24] and a $1/2$ approximation under the matroid constraint [15]. On the other hand, even for the cardinality constraint, the problem does not admit an approximation ratio better than $1 - e^{-1}$ [23]. It was a longstanding open question whether the problem admits a $(1 - e^{-1})$-approximation under the matroid constraint. In 2008, Vondrák [26] made a big breakthrough and answered this question affirmatively by proposing the so-called *continuous greedy* algorithm. Since then, plenty of optimal or well-performed approximation algorithms have been proposed for submodular maximization over a variety of constraints [2, 3, 4, 7, 12, 14, 16, 21, 22, 27].

For SMKP, a nearly optimal $(1-e^{-1}-\epsilon)$-approximation algorithm based on the continuous greedy technique was recently proposed in [9]. Their algorithm relies on two key ideas. First, they showed that by defining a configuration LP, an SMKP instance whose all bins have the same budget can be reduced to submodular maximization under 2-dimensional packing constraints (SMPC). Second, they developed a grouping technique inspired by [6] to convert a general SMKP instance to a leveled instance where bins are partitioned into blocks and bins in the same block have the same budget. In this way, they are able to reduce a general SMKP instance to an SMPC instance. They finally finished their work by a refined analysis of the continuous greedy algorithm for SMPC.

The techniques adopted by [9] and the way to combine them are somewhat sophisticated, which makes their algorithm not easy to understand and implement. Besides, the continuous greedy technique involves a sampling process and therefore their algorithm is inherently randomized. To the best of our knowledge, no deterministic algorithm was known for SMKP. In this paper, we present a simple deterministic combinatorial algorithm for SMKP, which achieves a $(1 - e^{-1} - \epsilon)$-approximation ratio.

▶ **Theorem 1.** *For any $\epsilon > 0$, there exists a deterministic combinatorial algorithm for SMKP that achieves a $(1 - e^{-1} - \epsilon)$-approximation ratio and runs in polynomial time.*

## 1.1 Technique Overview

We start with solving SMKP instances under the *identical case*, where all the bins have the same budget $B$. Such instances can be reduced to exponential-size instances of submodular maximization subject to a cardinality constraint. Inspired by this observation, we design an algorithm for the identical case by mimicking the greedy algorithm for the cardinality constraint. See Section 1.1.1 for details.

For any general SMKP instance, we use the grouping technique developed by [9] to convert it to the so-called leveled instance. While Fairstein et al. [9] resorts to the configuration LP to solve the leveled instance, we present a simple $(1 - e^{-1} - \epsilon)$-approximation algorithm for it by exploiting its structure and invoking our algorithm for the identical case as a subroutine. See Section 1.1.2 for details.

### 1.1.1 The Identical Case

Under the identical case, SMKP can be regarded as an exponential-size instance of submodular maximization subject to a cardinality constraint. Specifically, let $\mathcal{I} = \{S \subseteq N \mid c(S) \leq B\}$. For any $\mathcal{T} \subseteq \mathcal{I}$, define $g(\mathcal{T}) = f(\cup_{S \in \mathcal{T}} S)$. It is easy to verify that $g$ is a monotone submodular function. Then, $\max\{g(\mathcal{T}) \mid |\mathcal{T}| \leq m\}$ describes the SMKP instance under the identical case.

Inspired by the above observation, our algorithm packs bins one by one and manages to make each bin pack at least the average marginal value of the optimal solution over $m$ bins. In other words, for the $j$-th bin, it aims to find a set $S_j$ such that $f(S_j \mid \cup_{i=1}^{j-1} S_i) \geq \frac{1}{m} f(OPT \mid \cup_{i=1}^{j-1} S_i)$, where $OPT$ denotes the optimal solution. This naturally leads to $(1 - e^{-1})$ approximation.

We take the first bin as an example and explain that it is possible to find a set $S_1$ such that $f(S_1) \geq \frac{1}{m} f(OPT)$ when $m$ is large enough. If $S_1$ is obtained by packing elements in sequence greedily according to their marginal densities, then we can prove

$$f(S_1) \geq (1 - e^{-c(S_1)/c(OPT)}) \cdot f(OPT).$$

If we further allow $S_1$ to violate the budget constraint by adding one more element, then $c(S_1) \geq B$. Together with $c(OPT) \leq mB$, we have

$$f(S_1) \geq (1 - e^{-1/m}) \cdot f(OPT) \approx \frac{1}{m} f(OPT).$$

The story has not ended since the last element added to $S_1$ violates the budget constraint. To handle this issue, our algorithm divides elements into large and small elements according to their costs and then packs them in different ways. Specifically, an element $u \in N$ is large if $c(u) > \epsilon B$ and small otherwise. Our algorithm packs large elements by enumeration since there are polynomial ways to pack them in total. It packs small elements greedily as before. In this way, the last element added to $S_1$ has a cost less than $\epsilon B$ and there are at most $m$ such elements. Thus, all of them can be repacked using additional $\epsilon m$ bins and all $S_j$'s will then become feasible.

In Lemma 5, we show that $f(S_1) \geq \frac{1}{m} f(OPT)$ still holds although we introduce the enumeration step.

### 1.1.2 The General Case

Observe that a general SMKP instance can be reduced to an exponential-size instance of submodular maximization subject to a partition matroid constraint. Specifically, let $\mathcal{I}_j = \{S \subseteq N \mid c(S) \leq B_j\}$ be the feasible region for the $j$-th bin and $\mathcal{I} = \cup_{j=1}^{m} \mathcal{I}_j$. For any $\mathcal{T} \subseteq \mathcal{I}$, define $g(\mathcal{T}) = f(\cup_{S \in \mathcal{T}} S)$. Then, $\max\{g(\mathcal{T}) \mid |\mathcal{T} \cap \mathcal{I}_j| \leq 1, 1 \leq j \leq m\}$ describes the general SMKP instance. Recall that the optimal $(1 - e^{-1})$-approximation for the partition matroid constraint is obtained via the continuous greedy algorithm [26]. Thus, it is not a good idea to solve general SMKP instances directly.

The difficulty in solving general SMKP stems from that the budgets are distinct. Therefore, we first consider an "intermediate" instance where bins can be partitioned into $r$ blocks $\{M_k\}_{k=1}^{r}$ such that block $M_k$ contains sufficiently many bins and all of them have the same budget $B_k$. Clearly, this instance is slightly more general than the instance under the identical case. It can also be reduced to an exponential-size instance of submodular maximization subject to a partition matroid constraint. Specifically, let $\mathcal{I}_k = \{S \subseteq N \mid c(S) \leq B_k\}$ for $1 \leq k \leq r$ and $\mathcal{I} = \cup_{k=1}^{r} \mathcal{I}_k$. For any $\mathcal{T} \subseteq \mathcal{I}$, define $g(\mathcal{T}) = f(\cup_{S \in \mathcal{T}} S)$. Then, $\max\{g(\mathcal{T}) \mid |\mathcal{T} \cap \mathcal{I}_k| \leq |M_k|, 1 \leq k \leq r\}$ describes the above SMKP instance.

The above two reductions lead to different constraints $|\mathcal{T} \cap \mathcal{I}_j| \leq 1$ and $|\mathcal{T} \cap \mathcal{I}_k| \leq |M_k|$. For convenience, assume that $1/\epsilon$ is an integer, $|M_k| \geq 1/\epsilon$ and $\epsilon|M_k|$ is an integer for all $1 \leq k \leq r$. Our key observation is that for constraint $\{\mathcal{T} \subseteq \mathcal{I} \mid |\mathcal{T} \cap \mathcal{I}_k| \leq |M_k|, 1 \leq k \leq r\}$, there is a simple deterministic algorithm that can achieve $(1 - e^{-1} - \epsilon)$-approximation. The algorithm runs in $1/\epsilon$ iterations. In each iteration, block $M_k$ is visited in sequence and the algorithm will pack $\epsilon|M_k|$ bins in $M_k$. This forms an SMKP instance under the identical case. Thus, we can invoke our algorithm for the identical case to solve it.

Finally, we apply a grouping technique from [9] to convert a general instance to a $t$-leveled instance which has blocks $\{M_k\}_{k=1}^{r}$ and bins in the same block have the same budget. Besides, each of the first $t^2$ blocks contains a single bin, and each of the remaining blocks contains at least $t$. This is very similar to the intermediate instance before and it is not difficult to handle the first $t^2$ blocks.

## 1.2 Related Work

MKP has been fully studied previously. Kellerer [18] proposed the first PTAS for the identical case of the problem. Soon after, Chekuri and Khanna [5] proposed a PTAS for the general case. The result was later improved to an EPTAS by Jansen [17]. On the other hand, it is easy to see that the problem does not admit an FPTAS even for the case of $m = 2$ bins unless P = NP [5].

SMKP contains submodular maximization subject to a knapsack constraint as a special case. For this problem, there is an optimal $(1 - e^{-1})$-approximation algorithm that runs in $O(n^5)$ time [18, 25]. Later, a fast algorithm was proposed in [1] that achieves a $(1 - e^{-1} - \epsilon)$-approximation ratio and runs in $n^2(\log n/\epsilon)^{O(1/\epsilon^8)}$ time[2]. This was recently improved in [8] by a new algorithm that runs in $(1/\epsilon)^{O(1/\epsilon^4)} n \log^2 n$ time. The last two algorithms are impractical due to their high dependence on $1/\epsilon$. Very recently, a $(1 - e^{-1})$-approximation algorithm was proposed in [20, 13], which runs in $O(n^4)$ time. This algorithm can be further accelerated to achieve $(1 - e^{-1} - \epsilon)$-approximation in $\widetilde{O}(n^3/\epsilon)$ time.

To the best of our knowledge, SMKP was first considered in Feldman's Ph. D thesis [11]. Feldman proposed a polynomial time $(1/9 - o(1))$-approximation algorithm and a pseudo-polynomial time $1/4$ approximation algorithm for the general case of SMKP. For the identical case, he improved the results to a polynomial time $((e-1)/(3e-1) - o(1)) \approx 0.24$ approximation algorithm and a pseudo-polynomial time $(1 - e^{-1} - o(1))$-approximation algorithm. These algorithms are based on the *continuous greedy* technique and *contension resolution schemes* [27], and hence involve randomness inherently. Recently, Fairstein et al. [9] proposed a polynomial time randomized $(1 - e^{-1} - \epsilon)$-approximation algorithm for general SMKP.

## 1.3 Organization

In Section 2, we first formulate SMKP and introduce some notations. Then, we present a greedy algorithm that packs elements greedily according to their marginal densities. In Section 3, we present a $(1 - e^{-1} - \epsilon)$-approximation algorithm for SMKP under the identical case, assuming the number of bins $m \geq 1/(4\epsilon^3)$. In Section 4, we present a $(1 - e^{-1} - \epsilon)$-approximation algorithm for general SMKP. We conclude the paper and list some open problems in Section 5.

[2] As pointed out by [28, 8], the result in [1] has some issues.

**Algorithm 1** GREEDY.

**Input:** elements $N$, budgets $\{B_j\}_{j=1}^m$, profit $f$, cost $c$.
$S_j = \emptyset$ for $1 \le j \le m$ and $S = \cup_{j=1}^m S_j$.
**while** $N \setminus S \neq \emptyset$ *and there exists* $1 \le j \le m$ *such that* $c(S_j) < B_j$ **do**
  $u^* = \arg\max_{u \in N \setminus S} f(u \mid S)/c(u)$.
  $S_j = S_j + u^*$ and $S = S + u^*$.
**end**
**return** $S = \cup_{j=1}^m S_j$.

## 2 Preliminaries

An instance of the *submodular multiple knapsack problem* (SMKP) is defined as follows. We are given a set $N$ of $n$ elements and a set $M$ of $m$ bins (knapsacks). Each element $u \in N$ has a positive cost $c(u) > 0$. A subset $S \subseteq N$ of elements has a cost $c(S) = \sum_{u \in S} c(u)$. The $j$-th bin in $M$ has a positive budget $B_j > 0$ for $1 \le j \le m$. A subset $S \subseteq N$ is feasible for the problem if there is a disjoint partition $\{S_j\}_{j=1}^m$ of $S$ such that $c(S_j) \le B_j$ for $1 \le j \le m$. The profit of each subset $S \subseteq N$ of elements is specified by a *normalized*, *monotone* and *submodular* function $f : 2^N \to \mathbb{R}_+$. For a non-negative set function $f : 2^N \to \mathbb{R}_+$, it is called *normalized* if $f(\emptyset) = 0$, *monotone* if $f(S) \le f(T)$ for any $S \subseteq T$, and *submodular* if $f(S \cup \{u\}) - f(S) \ge f(T \cup \{u\}) - f(T)$ for any $S \subseteq T$ and $u \notin T$. The goal is to find a feasible set $S$ (and its partition $\{S_j\}_{j=1}^m$) such that the profit $f(S)$ (or $f(\cup_{j=1}^m S_j)$) is maximized.

An SMKP instance is specified by $(N, M, \{B_j\}_{j \in M}, f, c)$. Throughout this paper, we use $OPT$ to denote the optimal solution of an SMKP instance. Let $S + u$ be a shorthand for $S \cup \{u\}$. For the objective function $f$, we also use $f(u \mid S)$ and $f(T \mid S)$ to denote the marginal values $f(S + u) - f(S)$ and $f(S \cup T) - f(S)$, respectively. $f$ is accessed via a value oracle that returns $f(S)$ when set $S \subseteq N$ is queried. The query complexity of any algorithm for SMKP should be polynomial in the size of the problem.

### 2.1 The Greedy Algorithm

We first present a greedy algorithm, which is depicted as Algorithm 1. It serves as a cornerstone for other algorithms in this paper. It returns a (possibly infeasible) set with a $(1 - 1/e)$ approximation ratio. It packs elements one by one greedily, according to their *densities*, namely the ratios of their marginal values to their costs. The process continues provided there exists some bin whose budget has not been exhausted yet. As a side effect, each bin may pack one more element whose addition exceeds the budget of that bin. For convenience, we refer to this element as a *reserved element*. Nonetheless, we show that the set returned by Algorithm 1 has a large profit.

▶ **Lemma 2.** *Let $S$ be the set returned by Algorithm 1. For any set $X \subseteq N$, we have*

$$f(S) \ge \left(1 - e^{-c(S)/c(X)}\right) \cdot f(X).$$

**Proof.** If $c(S) < \sum_{j=1}^m B_j$, there is some $j$ such that $c(S_j) < B_j$. It means that Algorithm 1 ended with $S = N$. Thus, the lemma follows by monotonicity.

Now consider the case where $c(S) \ge \sum_{j=1}^m B_j$. Assume that $S = \{u_1, u_2, \ldots, u_\ell\}$, and for $0 \le i \le \ell$, $S^i = \{u_1, u_2, \ldots u_i\}$ denotes the first $i$ elements packed by Algorithm 1. Then, by the greedy rule,

$$\frac{f(u_i \mid S^{i-1})}{c(u_i)} \ge \frac{f(x \mid S^{i-1})}{c(x)}, \forall\, x \in X \setminus S^{i-1}.$$

By moving $c(x)$ to the left and summing over $x \in X \setminus S^{i-1}$,

$$c(X \setminus S^{i-1}) \cdot \frac{f(u_i \mid S^{i-1})}{c(u_i)} \geq \sum_{x \in X \setminus S^{i-1}} f(x \mid S^{i-1}) \geq f(X \setminus S^{i-1} \mid S^{i-1}).$$

The last inequality holds since $f$ is submodular. This gives us

$$\frac{f(S^i) - f(S^{i-1})}{c(u_i)} \geq \frac{f(X \setminus S^{i-1} \mid S^{i-1})}{c(X \setminus S^{i-1})} \geq \frac{f(X) - f(S^{i-1})}{c(X)}. \tag{1}$$

The last inequality holds since $f$ is monotone and $c(X \setminus S^{i-1}) \leq c(X)$.

Next, we assume that $f(X) > f(S^\ell)$, since otherwise the lemma already holds. Under this assumption, it must hold that $c(u_i) < c(X)$, since otherwise inequality (1) implies that $f(X) \leq f(S^i) \leq f(S^\ell)$. A contradiction! Now we can rearrange inequality (1) and obtain that

$$f(X) - f(S^i) \leq \left(1 - \frac{c(u_i)}{c(X)}\right)(f(X) - f(S^{i-1})).$$

By expanding the recurrence, we have

$$f(X) - f(S^i) \leq \prod_{j=1}^{i} \left(1 - \frac{c(u_j)}{c(X)}\right) \cdot f(X) \leq \prod_{j=1}^{i} e^{-\frac{c(u_j)}{c(X)}} \cdot f(X) = e^{-\frac{c(S^i)}{c(X)}} \cdot f(X).$$

The second inequality holds due to $e^x \geq 1 + x$. Hence we have

$$f(S^i) \geq \left(1 - e^{-c(S^i)/c(X)}\right) \cdot f(X).$$

The lemma follows by plugging $i = \ell$ into it. ◀

The above lemma immediately leads to the following corollary.

▶ **Corollary 3.** *The set $S$ returned by Algorithm 1 satisfies $f(S) \geq (1 - e^{-1}) \cdot f(OPT)$.*

**Proof.** If $c(S) < \sum_{j=1}^{m} B_j$, there is some $j$ such that $c(S_j) < B_j$. It means that Algorithm 1 ended with $S = N$. Thus, the corollary follows by monotonicity. If $c(S) \geq \sum_{j=1}^{m} B_j$, then $c(S) \geq c(OPT)$. The corollary follows from Lemma 2. ◀

## 3 The identical Case

In this section, we present a deterministic $(1 - e^{-1} - \epsilon)$ approximation algorithm for SMKP under the identical case, where all bins have the same budget. Our algorithm is depicted as Algorithm 2 and works when $m \geq 1/(4\epsilon^3)$. It packs bins one by one and manages to make each bin pack at least the average marginal value of the optimal solution over $m$ bins. In other words, for the $j$-th bin, it aims to find a set $S_j$ such that $f(S_j \mid \cup_{i=1}^{j-1} S_i) \geq \frac{1}{m} f(OPT \mid \cup_{i=1}^{j-1} S_i)$. This naturally leads to $(1 - e^{-1})$ approximation. For this purpose, Algorithm 2 divides elements into *large* and *small* elements according to their costs. Given input $\epsilon$, an element $u \in N$ is large if $c(u) > \epsilon B$ and small otherwise. Let $N_\ell = \{u \in N \mid c(u) > \epsilon B\}$ be the set of large elements and $N_s = N \setminus N_\ell$. For the $j$-th bin, Algorithm 2 first enumerates all feasible subsets of large elements. Then, for every such subset, Algorithm 1 is invoked over small elements to augment it. Finally, the one with the maximum marginal value is assigned to $S_j$.

Due to the call of Algorithm 1, $S_j$ might contain a reserved element, which is the last added into $S_j$ and violates the budget. To remedy this issue, Algorithm 2 divides the bins into two classes: the first $(1 - \epsilon)m$ bins are called *working bins* and the last $\epsilon m$ bins are

**Algorithm 2** IDENTICAL-CASE.

**Input:** elements $N$, budget $B$, number of bins $m$, profit $f$, cost $c$, constant $\epsilon > 0$.
1 Let the first $(1-\epsilon)m$ bins be *working bins* and the last $\epsilon m$ bins be *reserved bins*.
2 Define $N_\ell = \{u \in N \mid c(u) > \epsilon B\}$ and let $N_s = N \setminus N_\ell$.
3 $S_j = \emptyset$ for $1 \le j \le m$ and $T = \cup_{j=1}^{m} S_j$.
4 **for** $j = 1$ *to* $(1-\epsilon)m$ **do**
5  **foreach** *subset* $E \subseteq N_\ell$ *such that* $c(E) \le B$ **do**
6  $G_E = \text{GREEDY}(N_s, B - c(E), f(\cdot \mid T \cup E), c(\cdot))$.
7  **end**
8  $S_j = \arg\max_E f(E \cup G_E \mid T)$ and $T = \cup_{j=1}^{m} S_j$.
9 **end**
10 Repack the reserved elements in $T$ into the reserved bins.
11 **return** $T = \cup_{j=1}^{m} S_j$.

called *reserved bins*. The procedure described above only proceeds with the working bins. After that, Algorithm 2 repacks all reserved elements into the reserved bins. We will show that in this way, Algorithm 2 produces a feasible solution and the loss of the profit is little even if it does not use the reserved bins to pack new elements.

We now give an analysis of Algorithm 2. For $1 \le j \le (1-\epsilon)m$, let $S_j$ be defined as in line 8 of Algorithm 2 and $T_j = \cup_{i=1}^{j} S_i$. We first show that Algorithm 2 returns a feasible solution.

▶ **Lemma 4.** *Algorithm 2 produces a feasible solution.*

**Proof.** For $1 \le j \le (1-\epsilon)m$, observe that each $S_j$ contains at most one reserved element due to the call of Algorithm 1. By repacking those reserved elements into the reserved bins, each $S_j$ becomes feasible. Besides, the cost of each reserved element is at most $\epsilon B$ since it is a small element. Thus, a reserved bin can pack at least $1/\epsilon$ reserved elements. Then, $\epsilon m$ reserved bins can pack $m > (1-\epsilon)m$ reserved elements without exceeding their budgets. Therefore, Algorithm 2 produces a feasible solution. ◀

Next, we present Lemma 5 for Algorithm 2.

▶ **Lemma 5.** *Assume that* $m \ge 1/(4\epsilon^3)$. *For every* $1 \le j \le (1-\epsilon)m$,

$$f(S_j \mid T_{j-1}) \ge \frac{1-2\epsilon}{m} \cdot f(OPT \mid T_{j-1}).$$

**Proof.** For the sake of description, we define $g(\cdot) = f(\cdot \mid T_{j-1})$ and the lemma becomes $g(S_j) \ge \frac{1-2\epsilon}{m} \cdot g(OPT)$. Let $OPT_\ell = OPT \cap N_\ell$ and $OPT_s = OPT \setminus OPT_\ell$. We prove the lemma by case analysis, according to the cost and density of $OPT_s$.

**Case 1:** $c(OPT_s) \ge \epsilon m B$, namely $OPT_s$ has a large cost. Let $OPT_\ell = \cup_{j=1}^{m} OPT_{\ell,j}$ and $OPT_s = \cup_{j=1}^{m} OPT_{s,j}$, where $OPT_{\ell,j}$ and $OPT_{s,j}$ are the large and small elements packed in the $j$-th bin, respectively. For each $1 \le j \le m$, since $c(OPT_{\ell,j}) \le B$, $OPT_{\ell,j}$ will be enumerated during the **foreach** loop. Let $G_j$ be the output of GREEDY (Algorithm 1) starting from $OPT_{\ell,j}$. We will show that one of $OPT_{\ell,j} \cup G_j$ satisfies the lemma.
If $c(G_j) < B - c(OPT_{\ell,j})$, it means that Algorithm 1 ended with $G_j = OPT_s$ and therefore $g(G_j \mid OPT_{\ell,j}) = g(OPT_s \mid OPT_{\ell,j})$. If $c(G_j) \ge B - c(OPT_{\ell,j})$, then $c(G_j) \ge c(OPT_{s,j})$.

By Lemma 2,

$$\begin{aligned} g(G_j \mid OPT_{\ell,j}) &\geq \left(1 - e^{-c(G_j)/c(OPT_s)}\right) \cdot g(OPT_s \mid OPT_{\ell,j}) \\ &\geq \left(1 - e^{-c(OPT_{s,j})/c(OPT_s)}\right) \cdot g(OPT_s \mid OPT_{\ell,j}) \\ &\geq \left(\frac{c(OPT_{s,j})}{c(OPT_s)} - \frac{c(OPT_{s,j})^2}{2 \cdot c(OPT_s)^2}\right) \cdot g(OPT_s \mid OPT_{\ell,j}) \\ &\geq \left(\frac{c(OPT_{s,j})}{c(OPT_s)} - \frac{1}{2\epsilon^2 m^2}\right) \cdot g(OPT_s \mid OPT_{\ell,j}) \\ &\geq \left(\frac{c(OPT_{s,j})}{c(OPT_s)} - \frac{2\epsilon}{m}\right) \cdot g(OPT_s \mid OPT_{\ell,j}). \end{aligned}$$

The third inequality holds since $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$. The fourth inequality holds since $c(OPT_{s,j})/c(OPT_s) \leq 1/(\epsilon m)$. The last inequality holds since $m \geq 1/(4\epsilon^3)$. By adding $g(OPT_{\ell,j})$ on both sides of the last inequality and summing over $j$,

$$\begin{aligned} \sum_{j=1}^{m} g(OPT_{\ell,j} \cup G_j) &\geq \sum_{j=1}^{m} \left(\frac{c(OPT_{s,j})}{c(OPT_s)} - \frac{2\epsilon}{m}\right) \cdot g(OPT_s \mid OPT_{\ell,j}) + \sum_{j=1}^{m} g(OPT_{\ell,j}) \\ &\geq \sum_{j=1}^{m} \left(\frac{c(OPT_{s,j})}{c(OPT_s)} - \frac{2\epsilon}{m}\right) \cdot g(OPT_s \mid OPT_\ell) + g(OPT_\ell) \\ &= (1 - 2\epsilon) \cdot g(OPT_s \mid OPT_\ell) + g(OPT_\ell) \\ &\geq (1 - 2\epsilon) \cdot g(OPT). \end{aligned}$$

Hence, the maximum of $OPT_{\ell,j} \cup G_j$ satisfies the lemma and so does $S_j$.

**Case 2:** $g(OPT_s) \geq (1 - e^{-B/c(OPT_s)})^{-1} \cdot \frac{g(OPT)}{m}$, namely the density of $OPT_s$ is large. Consider one of the iterations of **foreach** loop where $E = \emptyset$. Note that it is augmented by $G_\emptyset$ via Greedy (Algorithm 1). If $c(G_\emptyset) < B$, it means that Algorithm 1 ended with $G_\emptyset = OPT_s$. Then,

$$g(G_\emptyset) = g(OPT_s) \geq (1 - e^{-B/c(OPT_s)})^{-1} \cdot \frac{g(OPT)}{m} \geq \frac{g(OPT)}{m}.$$

If $c(G_\emptyset) \geq B$, by Lemma 2,

$$g(G_\emptyset) \geq \left(1 - e^{-B/c(OPT_s)}\right) \cdot g(OPT_s) \geq \frac{g(OPT)}{m}.$$

This implies that $G_\emptyset$ satisfies the lemma and so does $S_j$.

**Case 3:** $c(OPT_s) < \epsilon m B$ and $g(OPT_s) < \left(1 - e^{-B/c(OPT_s)}\right)^{-1} \cdot \frac{g(OPT)}{m}$, namely both the cost and density of $OPT_s$ are small. We show that $OPT_s$ only contributes a negligible value in $OPT$:

$$\begin{aligned} g(OPT_s) &< (1 - e^{-1/\epsilon m})^{-1} \cdot \frac{g(OPT)}{m} \leq \left(\frac{1}{\epsilon m} - \frac{1}{2\epsilon^2 m^2}\right)^{-1} \frac{g(OPT)}{m} \\ &\leq \left(\frac{1}{2\epsilon m}\right)^{-1} \frac{g(OPT)}{m} = 2\epsilon \cdot g(OPT). \end{aligned}$$

The first inequality holds since $(1 - e^{-B/x})^{-1}$ is monotone increasing. The second holds since $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$. The third holds as long as $m \geq 1/\epsilon$. Hence, by submodularity,

$$g(OPT_\ell) \geq g(OPT) - g(OPT_s) \geq (1 - 2\epsilon) \cdot g(OPT),$$

and

$$\frac{1}{m}\sum_{j=1}^{m} g(OPT_{\ell,j}) \geq \frac{1}{m}\cdot g(OPT_\ell) \geq \frac{1-2\epsilon}{m}\cdot g(OPT).$$

This implies that the maximum of $OPT_{\ell,j}$ satisfies the lemma and so does $S_j$. ◀

By expanding the recurrence in Lemma 5, we have

▶ **Lemma 6.** *Assume that $m \geq 1/(4\epsilon^3)$. For every $1 \leq j \leq (1-\epsilon)m$,*

$$f(T_j) \geq (1 - e^{-j(1-2\epsilon)/m}) \cdot f(OPT).$$

**Proof.** By Lemma 5, for $1 \leq j \leq (1-\epsilon)m$,

$$f(S_j \mid T_{j-1}) \geq \frac{1-2\epsilon}{m}\cdot f(OPT \mid T_{j-1}).$$

By monotonicity of $f$,

$$f(T_j) - f(T_{j-1}) \geq \frac{1-2\epsilon}{m}\cdot (f(OPT) - f(T_{j-1})).$$

By rearranging the above inequality,

$$\left(1 - \frac{1-2\epsilon}{m}\right)(f(OPT) - f(T_{j-1})) \geq f(OPT) - f(T_j).$$

By expanding the recurrence,

$$f(OPT) - f(T_j) \leq \left(1 - \frac{1-2\epsilon}{m}\right)^j f(OPT) \leq e^{-j(1-2\epsilon)/m}\cdot f(OPT).$$

The last inequality holds since $e^{-x} \geq 1 - x$. Thus, we have

$$f(T_j) \geq (1 - e^{-j(1-2\epsilon)/m}) \cdot f(OPT).$$ ◀

We now provide a theoretical guarantee for Algorithm 2.

▶ **Theorem 7.** *When $m \geq 1/(4\epsilon^3)$, Algorithm 2 achieves a $(1 - e^{-1} - O(\epsilon))$ approximation ratio and uses $O(mn^{3+1/\epsilon})$ queries.*

**Proof.** For the approximation ratio, by plugging $j = (1-\epsilon)m$ into Lemma 6,

$$f(T_{(1-\epsilon)m}) \geq (1 - e^{-(1-\epsilon)(1-2\epsilon)}) \cdot f(OPT).$$

For the query complexity, observe that during the **foreach** loop, the number of subsets $E \subseteq N_\ell$ such that $c(E) \leq B$ is at most

$$\sum_{i=0}^{1/\epsilon} \binom{n}{i} = O(n^{1/\epsilon+1}).$$

Since each $E$ is augmented via Greedy, which uses $O(n^2)$ queries, the **foreach** loop uses $O(n^{1/\epsilon+3})$ in total. Then, Algorithm 2 overall uses $O(mn^{3+1/\epsilon})$ queries. ◀

# 4 The General Case

In this section, we present a deterministic $(1 - e^{-1} - \epsilon)$ approximation algorithm for solving general SMKP instances. A key difficulty is that the budgets of bins are distinct, which makes our technique for the identical case inapplicable. In Section 4.1, we introduce a grouping technique from [9], which reshapes any SMKP instance such that bins can be partitioned into *blocks* and almost every block contains sufficiently many bins with the same budget. Next, in Section 4.2, we show how one can design a nearly optimal algorithm for such instances.

## 4.1 Reshape the Instance

We first introduce a grouping technique from [9] to reshape any SMKP instance as follows.

▶ **Definition 8.** *A subset of bins $M' \subseteq M$ is called a block if for any $i, j \in M'$, $B_i = B_j$.*

▶ **Definition 9.** *For any $t \in \mathbb{N}_+$, a partition $\{M_k\}_{k=1}^r$ of bins $M$ is $t$-leveled if for every $1 \leq k \leq r$, $M_k$ is a block and $|M_k| = t^{\lfloor (k-1)/t^2 \rfloor}$.*

To gain some intuition, note that for every $1 \leq k \leq t^2$, block $M_k$ contains a single bin, and for every $t^2 < k \leq 2t^2$, block $M_k$ contains $t$ bins, etc. It follows that except for the first $t^2$ blocks, each of the remaining blocks contains at least $t$ bins with the same budget.

▶ **Lemma 10** ([9]). *There is a polynomial-time algorithm, referred to as* Block, *that takes a set of bins $M$, budgets $\{B_j\}_{j\in M}$ and a parameter $t \in \mathbb{N}_+$ as input, and returns a new set of bins $\widetilde{M} \subseteq M$, budgets $\{\widetilde{B}_j\}_{j\in\widetilde{M}}$ and a $t$-leveled partition $\{\widetilde{M}_k\}_{k=1}^r$ of bins $\widetilde{M}$ such that*

- *For every $j \in \widetilde{M}$, $\widetilde{B}_j \leq B_j$.*
- *For any SMKP instance $(N, M, \{B_j\}_{j\in M}, f, c)$ and a feasible solution $\{S_j\}_{j\in M}$ for it, there exists a feasible solution $\{\widetilde{S}_j\}_{j\in\widetilde{M}}$ for instance $(N, \widetilde{M}, \{\widetilde{B}_j\}_{j\in\widetilde{M}}, f, c)$ such that $f(\cup_{j\in\widetilde{M}} \widetilde{S}_j) \geq \left(1 - \frac{1}{t}\right) f(\cup_{j\in M} S_j)$ and $\cup_{j\in\widetilde{M}} \widetilde{S}_j \subseteq \cup_{j\in M} S_j$.*

The instance $(N, \widetilde{M}, \{\widetilde{B}_j\}_{j\in\widetilde{M}}, f, c)$ is called *$t$-leveled*. Lemma 10 tells us that any feasible solution for it is also feasible for the original instance $(N, M, \{B_j\}_{j\in M}, f, c)$, and an optimal solution for it causes a small loss in the profit.

## 4.2 The Final Algorithm

Now, we explain how one can design a nearly optimal algorithm for a $t$-leveled SMKP instance with bins $\widetilde{M}$, budgets $\{\widetilde{B}_j\}_{j\in\widetilde{M}}$ and a $t$-leveled partition $\{\widetilde{M}_k\}_{k=1}^r$ of $\widetilde{M}$.

For $t^2 < k \leq r$, block $\widetilde{M}_k$ contains $|\widetilde{M}_k| \geq t$ bins with the same budget $\widetilde{B}_k$. The problem restricted to each block $\widetilde{M}_k$ can be regarded as an SMKP instance under the identical case. Thus, a natural idea is to pack each block $\widetilde{M}_k$ in sequence by invoking Algorithm 2. However, we fail to get an optimal approximation via this procedure. Instead, we develop a technique that is inspired by [1]. We run $1/\epsilon$ iterations in total (assume that $1/\epsilon$ is an integer). In each iteration, we pack each block $\widetilde{M}_k$ in sequence but only pack $\epsilon|\widetilde{M}_k|$ bins (assume that $\epsilon|\widetilde{M}_k|$ is an integer). This forms an instance under the identical case with $\epsilon|\widetilde{M}_k|$ bins and therefore we can invoke Algorithm 2 to solve it.

For $1 \leq k \leq t^2$, block $\widetilde{M}_k$ contains a single bin with budget $\widetilde{B}_k$. Basically, we can use Greedy to pack elements. Likewise, we do not use the full budget at a time. Instead, we also run $1/\epsilon$ iterations. In each iteration, we pack elements using budget $(\epsilon - \epsilon^2)\widetilde{B}_k$. To avoid exceeding the budget, we only pack small elements $u$ satisfying $c(u) \leq \epsilon^2 \widetilde{B}_k$. To ensure this, we need to enumerate large-valued and large-cost elements in this bin. The overall procedure is depicted as Algorithm 3.

**Algorithm 3** The Final Algorithm for SMKP.

**Input:** elements $N$, bins $M$, budgets $\{B_j\}_{j\in M}$, profit $f$, cost $c$, constant $\epsilon > 0$.

Let $s = 1/(16\epsilon^9)$ and $t = 1/(4\epsilon^3)$.
Let $\mathcal{C} = \emptyset$.
$(\widetilde{M}, \{\widetilde{B}_j\}_{j\in\widetilde{M}}, \{\widetilde{M}_k\}_{k=0}^{r}) = \textsc{Block}(M, \{B_j\}_{j\in M}, t)$.
**foreach** *feasible solution* $\{E_j\}_{j=1}^{t^2}$ *such that* $|\cup_{j=1}^{m} E_j| \le s$ **do** \\ $E_j = \emptyset$ for $j > t^2$
  Let $E = \cup_{j=1}^{t^2} E_j$.
  Let $S_j = E_j$ for $1 \le j \le t^2$ and $S_j = \emptyset$ for $t^2 < j \le k$.
  **for** $i = 1$ *to* $1/\epsilon$ **do**
    **for** $k = 1$ *to* $r$ **do** \\ handle blocks one by one
      **if** $k \le t^2$ **then** \\ each block contains a single bin
        Let $D = \{u \in N \setminus E \mid f(u \mid E) > \frac{1}{s} \cdot f(E)\}$.
        Let $L_k = \{u \in N \setminus E \mid c(u) > \epsilon^2(\widetilde{B}_k - c(E_k))\}$.
        $R_k = \textsc{Greedy}(N \setminus (E \cup D \cup L_k), (\epsilon - \epsilon^2)(\widetilde{B}_k - c(E_k)), f(\cdot \mid \cup_{j=1}^{m} S_j), c(\cdot))$.
        $S_{k+1} = S_{k+1} \cup R_k$.
      **else** \\ each block contains $\ge t$ bins
        $\{R_j\}_{j\in\widetilde{M}_k} = \textsc{Identical-case}(N \setminus E, \widetilde{B}_k, \epsilon|\widetilde{M}_k|, f(\cdot \mid \cup_{j=1}^{m} S_j), c(\cdot), \epsilon)$.
        $S_j = S_j \cup R_j$ for $j \in \widetilde{M}_k$.
      **end**
    **end**
  **end**
  $\mathcal{C} = \mathcal{C} \cup \{\{S_j\}_{j=1}^{m}\}$.
**end**
**return** $\arg\max\{f(\cup_{j=1}^{m} S_j) \mid \{S_j\}_{j=1}^{m} \in \mathcal{C}\}$.

▶ **Theorem 11.** *Algorithm 3 achieves a* $1 - e^{-1} - O(\epsilon)$ *approximation ratio and uses a polynomial number of queries.*

**Proof.** Let $\widetilde{OPT} = \cup_{j=1}^{m} \widetilde{OPT}_j$ be the optimal solution of the SMKP instance with bins $\widetilde{M}$ and budgets $\{\widetilde{B}_j\}_{j\in\widetilde{M}}$. Let $OPT' = \cup_{j=1}^{t^2} \widetilde{OPT}_j$. Order elements in $OPT'$ greedily according to their marginal values such that $o_1 = \arg\max_{o\in OPT'} f(o)$, $o_2 = \arg\max_{o\in OPT'\setminus\{o_1\}} f(o \mid o_1)$, etc. Denote by $E$ the first $s$ elements in $OPT'$ (if $|OPT'| < s$, then $E = OPT'$). Let $E_j = E \cap \widetilde{OPT}_j$ for $1 \le j \le t^2$. Then, $\{E_j\}_{j=1}^{t^2}$ will be enumerated during the **foreach** loop. In the following, we focus on this particular set.

Let $D = \{u \in N \mid f(u \mid E) > \frac{1}{s} \cdot f(E)\}$. Since $E$ is the first $s$ elements in $OPT'$, we have $f(o \mid E) \le \frac{1}{s} \cdot f(E)$ for any $o \in OPT' \setminus E$. Thus, $D \cap (OPT' \setminus E) = \emptyset$ and therefore $OPT' \setminus E$ will not be excluded from the execution of $\textsc{Greedy}$ over $N \setminus (E \cup D)$. Besides, $\{\widetilde{OPT}_j \setminus E_j\}_{j=1}^{t^2}$ is a feasible solution given budgets $\{\widetilde{B}_j - c(E_j)\}_{j=1}^{t^2}$.

For $1 \le j \le t^2$, let $L_j = \{u \in N \setminus E \mid c(u) > \epsilon^2(\widetilde{B}_j - c(E_j))\}$. Define $OPT^*$ as follows. For $1 \le j \le t^2$, $OPT_j^* = \widetilde{OPT}_j \setminus L_j$. For $j > t^2$, $OPT_j^* = \widetilde{OPT}_j$. Then, $OPT^* = \cup_{j=1}^{m} OPT_j^*$. We have

$$\begin{aligned} f(OPT^* \mid E) &= f((\cup_{j=1}^{t^2} \widetilde{OPT}_j \setminus L_j) \cup (\cup_{j=t^2+1}^{m} \widetilde{OPT}_j) \mid E) \\ &\ge f(\cup_{j=1}^{m} \widetilde{OPT}_j \mid E) - f(\cup_{j=1}^{t^2} \widetilde{OPT}_j \cap L_j \mid E) \end{aligned}$$

$$\begin{aligned}
&\geq f(\widetilde{OPT} \mid E) - \sum_{j=1}^{t^2} \sum_{u \in (\widetilde{OPT}_j \setminus E) \cap L_j} f(u \mid E) \\
&\geq f(\widetilde{OPT} \mid E) - \frac{t^2}{\epsilon^2 s} \cdot f(E) \\
&= f(\widetilde{OPT} \mid E) - \epsilon \cdot f(E).
\end{aligned}$$

The first two inequalities are due to submodularity. The third inequality holds since by definition of $L_j$, $\widetilde{OPT}_j \setminus E$ contains at most $1/\epsilon^2$ elements in $L_j$, and $f(u \mid E) \leq \frac{1}{s} f(E)$ due to $D \cap (\widetilde{OPT}_j \setminus E) = \emptyset$. The last equality follows from the choices of $t$ and $s$. This implies that invoking Greedy over $N \setminus (E \cup D \cup L_j)$ for $1 \leq j \leq t^2$ only incurs little loss in the profit.

Now we are prepared to provide a theoretical bound for Algorithm 3. Let $g(\cdot) = f(\cdot \mid E)$. For $1 \leq i \leq 1/\epsilon$ and $1 \leq k \leq r$, let $R_{ik}$ be the set returned in line 12 if $k \leq t^2$ and $R_{ik} = \cup_{j \in \widetilde{M}_k} R_j$ otherwise, where $\{R_j\}_{j \in \widetilde{M}_k}$ is the set returned in line 15. Then, the $k$-th block $\widetilde{M}_k$ packs $\cup_{i=1}^{1/\epsilon} R_{ik}$ by the end of Algorithm 3. Define $T_0 = \emptyset$ and $T_i = T_{i-1} \cup (\cup_{k=1}^{r} R_{ik})$ for $1 \leq i \leq 1/\epsilon$.

For $1 \leq i \leq 1/\epsilon$ and $1 \leq k \leq t^2$, by Lemma 2,

$$\begin{aligned}
g(R_{ik} \mid T_{i-1} \cup (\cup_{k'=1}^{k-1} R_{ik'})) &\geq (1 - e^{-(\epsilon - \epsilon^2)}) \cdot g(OPT_k^* \mid T_{i-1} \cup (\cup_{k'=1}^{k-1} R_{ik'})) \\
&\geq (\epsilon - 2\epsilon^2) \cdot g(OPT_k^* \mid T_i).
\end{aligned}$$

The last inequality holds due to $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$ and submodularity.

For $1 \leq i \leq 1/\epsilon$ and $t^2 < k \leq r$, by Lemma 6,

$$\begin{aligned}
g(R_{ik} \mid T_{i-1} \cup (\cup_{k'=1}^{k-1} R_{ik'})) &\geq (1 - e^{-\epsilon(1-2\epsilon)}) \cdot g(OPT_k^* \mid T_{i-1} \cup (\cup_{k'=1}^{k-1} R_{ik'})) \\
&\geq (\epsilon - 3\epsilon^2) \cdot g(OPT_k^* \mid T_i).
\end{aligned}$$

Again, the last inequality holds due to $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$ and submodularity.

Summing up over $1 \leq k \leq r$, we have

$$\begin{aligned}
g(T_i) - g(T_{i-1}) &= \sum_{k=1}^{r} g(R_{ik} \mid T_{i-1} \cup (\cup_{k'=1}^{k-1} R_{ik'})) \\
&\geq \sum_{k=1}^{r} (\epsilon - 3\epsilon^2) \cdot g(OPT_k^* \mid T_i) \\
&\geq (\epsilon - 3\epsilon^2) \cdot g(OPT^* \mid T_i) \\
&\geq (\epsilon - 3\epsilon^2) \cdot (g(OPT^*) - g(T_i)).
\end{aligned}$$

The last two inequalities are due to submodularity and monotonicity, respectively. By adding $g(OPT^*)$ to both sides and move $g(T_i)$ to the right in the above inequality,

$$g(OPT^*) - g(T_{i-1}) \geq (1 + \epsilon - 3\epsilon^2)(g(OPT^*) - g(T_i)).$$

This leads to

$$g(OPT^*) - g(T_i) \leq \frac{1}{(1 + \epsilon - 3\epsilon^2)^i} \cdot g(OPT^*).$$

Hence, by plugging $i = 1/\epsilon$,

$$\begin{aligned}
g(T_{1/\epsilon}) &\geq \left(1 - \frac{1}{(1 + \epsilon - 3\epsilon^2)^{1/\epsilon}}\right) \cdot g(OPT^*) = \left(1 - e^{-\frac{1}{\epsilon} \ln(1 + \epsilon - 3\epsilon^2)}\right) \cdot g(OPT^*) \\
&\geq (1 - e^{-\frac{1}{\epsilon}(\epsilon - 3\epsilon^2 - (\epsilon - 3\epsilon^2)^2/2)}) \cdot g(OPT^*) \geq (1 - e^{-1} - O(\epsilon)) \cdot g(OPT^*).
\end{aligned}$$

The second inequality holds since $\ln(1+x) \geq x - x^2/2$ for $x > 0$. Finally, recall that $g(\cdot) = f(\cdot \mid E)$, we have

$$\begin{aligned} f(T_{1/\epsilon}) &= f(E) + f(T_{1/\epsilon} \mid E) \geq f(E) + (1 - e^{-1} - O(\epsilon)) \cdot f(OPT^* \mid E) \\ &\geq f(E) + (1 - e^{-1} - O(\epsilon)) \cdot (f(\widetilde{OPT} \mid E) - \epsilon f(E)) \\ &\geq (1 - e^{-1} - O(\epsilon)) \cdot f(OPT). \end{aligned}$$

◀

## 5 Conclusion

In this paper, we present a deterministic $(1 - e^{-1} - \epsilon)$-approximation algorithm for SMKP. Our algorithm is inspired by the viewpoint regarding SMKP instances as exponential-size instances of submodular maximization subject to a cardinality or partition matroid constraint. Thus our algorithm is conceptually much simpler than that of Fairstein et al. [9].

As pointed out by [9], it remains open to remove the loss of $\epsilon$ in the approximation ratio. As a first step, we present a $(1 - e^{-1})$-approximation algorithm for SMKP when the number of bins $m$ is constant in the full version of this paper. Recently, a randomized 0.385-approximation algorithm for *non-monotone* SMKP was proposed in [10]. It is an interesting question to design deterministic algorithms for this problem.

## References


1. Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1497–1514, 2014. doi:10.1137/1.9781611973402.110.
2. Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 392–403, 2016. doi:10.1137/1.9781611974331.ch29.
3. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 649–658, 2012. doi:10.1109/FOCS.2012.73.
4. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1433–1452, 2014. doi:10.1137/1.9781611973402.106.
5. Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA.*, pages 213–222, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338254.
6. Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981. doi:10.1007/BF02579456.
7. Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 248–257, 2016. doi:10.1109/FOCS.2016.34.
8. Alina Ene and Huy L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 53:1–53:12, 2019. doi:10.4230/LIPIcs.ICALP.2019.53.

9 Yaron Fairstein, Ariel Kulik, Joseph (Seffi) Naor, Danny Raz, and Hadas Shachnai. A ($1-e^{-1}-\epsilon$)-approximation for the monotone submodular multiple knapsack problem. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 44:1–44:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.44.

10 Yaron Fairstein, Ariel Kulik, and Hadas Shachnai. Modular and submodular optimization with multiple knapsack constraints via fractional grouping. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.41.

11 Moran Feldman. *Maximization problems with submodular objective functions*. Technion-Israel Institute of Technology, Faculty of Computer Science, 2013.

12 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 570–579, 2011. doi:10.1109/FOCS.2011.46.

13 Moran Feldman, Zeev Nutov, and Elad Shoham. Practical budgeted submodular maximization. *Algorithmica*, 85(5):1332–1371, 2023. doi:10.1007/s00453-022-01071-2.

14 Yuval Filmus and Justin Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 659–668, 2012. doi:10.1109/FOCS.2012.55.

15 Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions - II. In *Polyhedral combinatorics*, pages 73–87. Springer, 1978.

16 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1098–1116, 2011. doi:10.1137/1.9781611973082.83.

17 Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 665–674, 2009. doi:10.1137/1.9781611973068.73.

18 Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Randomization, Approximation, and Combinatorial Algorithms and Techniques, RANDOM-APPROX'99, Berkeley, CA, USA, August 8-11, 1999, Proceedings*, pages 51–62, 1999. doi:10.1007/978-3-540-48413-4_6.

19 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 137–146. ACM, 2003. doi:10.1145/956750.956769.

20 Ariel Kulik, Roy Schwartz, and Hadas Shachnai. A refined analysis of submodular greedy. *Oper. Res. Lett.*, 49(4):507–514, 2021. doi:10.1016/j.orl.2021.04.006.

21 Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 323–332. ACM, 2009. doi:10.1145/1536414.1536459.

22 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010. doi:10.1287/moor.1100.0463.

**23** George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978. doi:10.1287/moor.3.3.177.

**24** George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

**25** Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004. doi:10.1016/S0167-6377(03)00062-2.

**26** Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74, 2008. doi:10.1145/1374376.1374389.

**27** Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 783–792, 2011. doi:10.1145/1993636.1993740.

**28** Yuichi Yoshida. Maximizing a monotone submodular function with a bounded curvature under a knapsack constraint. *SIAM J. Discret. Math.*, 33(3):1452–1471, 2019. doi:10.1137/16M1107644.

# The Tight Spanning Ratio of the Rectangle Delaunay Triangulation


**André van Renssen**
University of Sydney, Australia

**Yuan Sha**
University of Sydney, Australia

**Yucheng Sun**
University of Sydney, Australia

**Sampson Wong**
BARC, University of Copenhagen, Denmark



## Abstract

Spanner construction is a well-studied problem and Delaunay triangulations are among the most popular spanners. Tight bounds are known if the Delaunay triangulation is constructed using an equilateral triangle, a square, or a regular hexagon. However, all other shapes have remained elusive. In this paper we extend the restricted class of spanners for which tight bounds are known. We prove that Delaunay triangulations constructed using rectangles with aspect ratio $A$ have spanning ratio at most $\sqrt{2}\sqrt{1+A^2+A\sqrt{A^2+1}}$, which matches the known lower bound.



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

**Keywords and phrases** Spanners, Delaunay Triangulation, Spanning Ratio

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.99

**Related Version** *Full Version*: https://arxiv.org/abs/2211.11987


## 1 Introduction

A geometric graph is a weighted graph in the plane where every vertex $v$ has coordinates $(x_v, y_v)$ and the weight of an edge between any two vertices is the Euclidean distance between its endpoints. A *geometric spanner* is defined to be a class of subgraphs where the shortest path distance between any two vertices is at most the Euclidean distance between these two vertices multiplied by a constant $t$. The smallest constant $t$ for which this property holds is called the *spanning ratio* or *stretch factor* of the geometric spanner. A comprehensive overview on the topic of geometric spanners can be found in the book by Narasimhan and Smid [11] and the survey by Bose and Smid [5].

One way to construct a geometric spanner is by using a Delaunay triangulation. The Delaunay triangulation is defined as follows: for any two vertices $u$ and $v$, if there exists a circle with $u$ and $v$ on its boundary and no other vertex in its interior, then the edge between $u$ and $v$ is part of the Delaunay triangulation. Equivalently, this can be defined using three vertices $u$, $v$, and $w$, where the triangle connecting these three vertices is part of the Delaunay triangulation if and only if the unique circle through these three points does not contain any other vertices in its interior. For simplicity, it is usually assumed that no three points are collinear and no four points lie on the boundary of the circle.

The tight spanning ratio of the Delaunay triangulation is not known. Dobkin et al. [8] showed an upper bound of $\pi(1+\sqrt{5})/2 \approx 5.09$ for the spanning ratio, which Keil and Gutwin [9] improved to $4\pi/3\sqrt{3} \approx 2.42$. Currently, the best upper bound is 1.998, proven by Xia [14]. A lower bound on the spanning ratio was provided by Bose et al. [4], who showed that this is strictly larger than $\frac{\pi}{2}$. This was later improved to 1.59 by Xia and Zhang [15].

Usually, the distance between two points $u$ and $v$ in the plane is defined as $((x_u - x_v)^2 + (y_u - y_v)^2)^{\frac{1}{2}}$. This distance can be generalized to a family of metrics $L_p$ where the distance between $u$ and $v$ is defined as $((x_u - x_v)^p + (y_u - y_v)^p)^{\frac{1}{p}}$. The shape of a "circle" varies in different metrics, leading to different Delaunay triangulations in different metrics. For example, the shape of the "circle" would be a diamond or square in the $L_1$ and $L_\infty$ metrics.

In 1986, Lee and Lin [10] introduced the notion of generalized Delaunay triangulations. Instead of using a circle to construct the graph, generalized Delaunay triangulations can be constructed using arbitrary geometric shapes. It was proven that any generalized Delaunay triangulation constructed using a convex shape is a spanner [2].

Although generalized Delaunay triangulations using arbitrary convex shapes are known to be spanners, their spanning ratios are less well understood. Tight bounds on the spanning ratio are only known when an equilateral triangle, a square, or a regular hexagon is used in the construction. When using equilateral triangles, Chew [7] showed that the spanning ratio is 2. When using squares, Chew [6] showed an upper bound of $\sqrt{10} \approx 3.16$, and Bonichon et al. [1] showed a matching upper and lower bound of $\sqrt{4 + 2\sqrt{2}} \approx 2.61$. When using regular hexagons, Perković et al. [12] showed a tight bound of 2.

Bose et al. [3] studied generalized Delaunay triangulations using rectangles. For rectangles with aspect ratio $A$, they showed an upper bound of $\sqrt{2}(2A + 1)$ and a lower bound of $\sqrt{2}\sqrt{1 + A^2 + A\sqrt{A^2 + 1}}$. Inspired by the proof of Bonichon et al. [1], by significantly extending and generalizing their approach we obtain a tight bound of $\sqrt{2}\sqrt{1 + A^2 + A\sqrt{A^2 + 1}}$. This extends the class of shapes for which a tight bound is known for the spanning ratio of generalized Delaunay triangulations. We note that the proof of our result is not a straightforward extension of Bonichon et al. [1], as we cannot simply rotate our lemmas to get them to prove both the horizontal and vertical cases simultaneously.

## 2 Preliminaries

Let us first formally define the rectangle Delaunay triangulation of a set of points $P$. Given an arbitrary axis-aligned rectangle $R$, the rectangle Delaunay triangulation is constructed by considering scaled translates of $R$ (rotations are not allowed). Such scaled translates are also referred to as homothets. Given two vertices $u$ and $v$ in $P$, the rectangle Delaunay triangulation contains an edge between $u$ and $v$ if and only if there exists a scaled translate of $R$ with $u$ and $v$ on its boundary which contains no vertices of $P$ in the interior. Equivalently, the rectangle Delaunay triangulation contains a triangle $\triangle uvw$ if and only if there exists a scaled translate of $R$ with $u$, $v$, and $w$ on its boundary which contains no vertices of $P$ in the interior. We note that different rectangles can give different rectangle Delaunay triangulations.

For our proofs, we assume that $P$ is in general position. Specifically, we assume that no four vertices lie on the boundary of any scaled translate of $R$ and that no two vertices lie on a line parallel to any of the sides of $R$ (i.e., no two vertices lie on a vertical or horizontal line). These assumptions are common for Delaunay graphs and are required to guarantee their planarity.

Throughout this paper, we use $A$ to denote the aspect ratio of the rectangle $R$ used in the construction of the rectangle Delaunay triangulation, i.e., $A = l/s$ where $l$ and $s$ are the length of the long and short side of $R$ respectively. We also use $d_t(u, v)$ to denote the length of the shortest path in the rectangle Delaunay triangulation between $u$ and $v$, $d_x(u, v)$ to denote the difference in $x$-coordinate between $u$ and $v$, $d_y(u, v)$ to denote the difference in $y$-coordinate between $u$ and $v$, and $d_2(u, v)$ to denote the Euclidean distance between $u$ and $v$.

Due to space constraints, we have deferred proofs of several lemmas to the full version of this paper [13].

## 3 Bounding the Spanning Ratio

To show an upper bound on the spanning ratio between any two vertices $u$ and $v$, we consider the sequence of triangles $T_1, T_2, ..., T_k$ intersecting with line segment $uv$. The order of this sequence is determined by the order in which these triangles are encountered when following $uv$ from $u$ to $v$ (as shown in Figure 1). Each triangle except $T_1$ and $T_k$ intersects the interior of $uv$ twice. Hence, we can define the last line segment of $T_i$ $(1 \leq i < k)$ that intersects $uv$ as the line segment involved in the second intersection. We use $h_i$ and $l_i$ to denote the endpoints of the last line segment of $T_i$, where $h_i$ is the endpoint above $uv$ and $l_i$ is the endpoint below $uv$. Since all $T_i$ are triangles, we have that for every $T_i$ and $T_{i+1}$, either $l_i = l_{i+1}$ or $h_i = h_{i+1}$. We also define $h_0 = l_0 = u$, $l_k = h_k = v$.

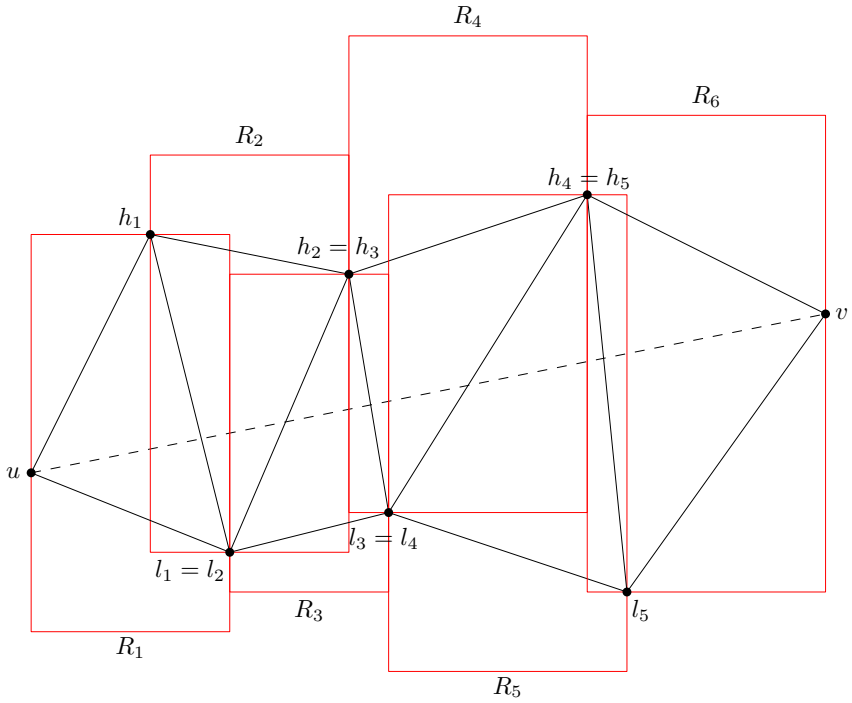



**Figure 1** The triangles intersecting $uv$ and their associated rectangles and $h_i$ and $l_i$.

Each triangle $T_i$ also has an associated rectangle $R_i$: the scaled translate of $R$ that has the three vertices of $T_i$ on its boundary. For ease of reference, we use W (west), N (north), E (east), and S (south) to refer to the four sides of a rectangle. We also use these sides to classify an edge, for example, if an endpoint of an edge lies on the W side of $R_i$ and the other endpoint lies on the N side of $R_i$, we call the edge a WN edge. We also define $u$ to be on the E side of $R_0$ (not associated with any triangle), as this will simplify some of the lemma statements.

Define $L$ to be the length of the vertical side of $R$ divided by the length of the horizontal size of $R$. Note that $L$ can be either $A$ or $1/A$. For our proofs, it is helpful to distinguish between edges of slope less than the slope of the diagonal of $R$ and those with larger slope.

▶ **Definition 1.** *An edge is gentle if it has a slope within [-L, L]. Otherwise it is steep.*

We let $u, v$ be any two vertices in the rectangle Delaunay triangulation. Fix the $(x, y)$-coordinate system so that we have $Ld_x(u, v) > d_y(u, v)$. Note that this is without loss of generality, since we can simply switch the $x$- and $y$-axes if needed. This implies that if we consider a scaled translate of $R$ with $u$ lying in the lower left corner and passing through $v$, then $v$ lies on the E side. Without loss of generality, we assume $u$ to be at the origin $(0, 0)$ and $v$ to be at $(x, y)$. We use $R(u, v)$ to denote the rectangle with $u$ and $v$ in opposite corners.

In order to bound the spanning ratio of the rectangle Delaunay triangulation, we first define what it means for a rectangle to have potential. This later helps us to bound the total length of the shortest path between $u$ and $v$ in the rectangle Delaunay triangulation.

▶ **Definition 2.** *The inductive point $c$ of a rectangle $R_i$ is the point with larger $x$-coordinate out of $h_i$ and $l_i$. Rectangle $R_i$ is inductive if edge $(l_i, h_i)$ is gentle.*

▶ **Definition 3.** *A rectangle $R_i$ has potential if $d_t(u, h_i) + d_t(u, l_i) + d_{R_i}(h_i, l_i) \leq (2 + 2L)x_i$ where $d_{R_i}(h_i, l_i)$ is the Euclidean distance when moving clockwise from $h_i$ to $l_i$ along the sides of $R_i$ and $x_i$ is the $x$-coordinate of the E side of $R_i$.*

We are now ready to prove that rectangles that are not inductive pass on their potential.

▶ **Lemma 4.** *If $R(u, v)$ is empty and $(u, v)$ is not an edge in the rectangle Delaunay triangulation, then $R_1$ has potential. Furthermore, for any $1 \leq i < k$, if $R_i$ has potential but is not inductive, then $R_{i+1}$ has potential.*

Next, we bound the distance from $u$ to the inductive point of a rectangle with potential when this inductive point lies on the E side of the rectangle.

▶ **Lemma 5.** *If rectangle $R_i$ has potential and its inductive point $c$ ($c = h_i$ or $c = l_i$) lies on the E side of $R_i$, then $d_t(u, c) \leq (1 + L)x_c$.*

Now we shift our focus to paths consisting of gentle edges (see Figure 2).

▶ **Definition 6.** *If $h_j$ is on the E side of $R_j$, the maximal high path ending at $h_j$ is $h_j$ itself; otherwise, it is the path $h_i, h_{i+1}, ..., h_j$ such that $h_m$ is not on the E side of $R_m$ (for $i < m \leq j$) and either $i = 0$ or $h_i$ is on the E side of $R_i$.*

*If $l_j$ is on the E side of $R_j$, the maximal low path ending at $l_j$ is $l_j$; otherwise, it is the path $l_i, l_{i+1}, ..., l_j$ such that $l_m$ is not on the E side of $R_m$ (for $i < m \leq j$) and either $i = 0$ or $l_i$ is on the E side of $R_i$.*

Next, we bound the length of these maximal high and maximal low paths.

▶ **Lemma 7.** *If the path $h_i, h_{i+1}, ..., h_j$ is a maximal high path then $d_t(h_i, h_j) \leq (x_{h_j} - x_{h_i}) + (y_{h_j} - y_{h_i})$. Similarly, if the path $l_i, l_{i+1}, ..., l_j$ is a maximal low path then $d_t(l_i, l_j) \leq (x_{l_j} - x_{l_i}) + (y_{l_i} - y_{l_j})$.*

We now use the above lemmas to prove bounds on the path length from $u$ to the inductive point on the first inductive rectangle (if one exists) when $R(u, v)$ does not contain any vertices. Note that in Property 2 of Lemma 8, we differentiate between $L = A$ and $L = 1/A$, which is crucial in proving Theorem 10.

▶ **Lemma 8.** *Let $R(u, v)$ not contain any vertices of $P$ and let $(u, v)$ not be an edge of the rectangle Delaunay triangulation. The following properties hold:*

**1.** *If no rectangle in $R_1, ..., R_k$ is inductive then*

$$d_t(u, v) \leq (L + \sqrt{L^2 + 1})x + y.$$

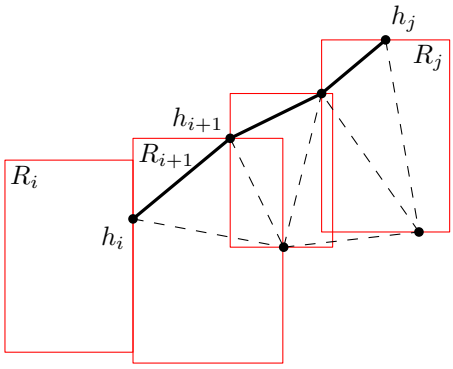



**Figure 2** An example of a maximal high path (thick edges). The other edges of the triangles are shown using dashed line segments.

2. *Otherwise, let $R_j$ be the first inductive rectangle in the sequence $R_1, ..., R_k$.*
   a. *If $h_j$ is the inductive point of $R_j$ and $L = A$, then*

$$d_t(u, h_j) + (y_{h_j} - y) \leq (A + \sqrt{A^2 + 1})x_{h_j}.$$

   b. *If $h_j$ is the inductive point of $R_j$ and $L = \frac{1}{A}$, then*

$$d_t(u, h_j) + A(y_{h_j} - y) \leq \left(1 + \sqrt{\tfrac{1}{A^2} + 1}\right) x_{h_j}.$$

   c. *If $l_j$ is the inductive point of $R_j$ and $L = A$, then*

$$d_t(u, l_j) - y_{l_j} \leq (A + \sqrt{A^2 + 1})x_{l_j}.$$

   d. *If $l_j$ is the inductive point of $R_j$ and $L = \frac{1}{A}$, then*

$$d_t(u, l_j) - Ay_{l_j} \leq \left(1 + \sqrt{\tfrac{1}{A^2} + 1}\right) x_{l_j}.$$

**Proof.** *Property 1:* By Lemma 4, if no rectangle in $R_1, ..., R_k$ is inductive then the last rectangle must have potential since $R_1$ has potential. Since no two vertices have the same $y$-coordinate, $v$ must lie on the E side of the last rectangle. Thus, we can use Lemma 5 to conclude that $d_t(u, v) \leq (1 + L)x \leq (L + \sqrt{L^2 + 1})x + y$.

*Property 2a:* We consider the situation where $R_j$ is the first inductive rectangle in the sequence $R_1, ..., R_k$. Let $l_i, ..., l_{j-1} = l_j$ be the maximal low path ending at $l_j$, and recall that $h_j$ is the inductive point of $R_j$. By Lemma 4 we know that $R_i$ has potential, since $R_1$ has potential and no rectangle before $R_i$ is inductive. Since $R_i$ has potential and $l_i$ is on the E side of $R_i$, by Lemma 5 we know $d_t(u, l_i) \leq (1 + L)x_{l_i}$. See Figure 3. Since $L = A$, we have

$$\begin{aligned} d_t(u, h_j) + (y_{h_j} - y) &\leq d_t(u, l_i) + d_t(l_i, l_j) + d_2(l_j, h_j) + (y_{h_j} - y) \\ &\leq (1 + A)x_{l_i} + d_t(l_i, l_j) + d_2(l_j, h_j) + y_{h_j}. \end{aligned}$$

Since $l_i, ..., l_{j-1} = l_j$ is a maximal low path, by Lemma 7 we know $d_t(l_i, l_j) \leq (x_{l_j} - x_{l_i}) + (y_{l_i} - y_{l_j})$. Hence, we obtain that:

$$\begin{aligned} d_t(u, h_j) + (y_{h_j} - y) &\leq (1 + A)x_{l_i} + (x_{l_j} - x_{l_i}) + (y_{l_i} - y_{l_j}) + d_2(l_j, h_j) + y_{h_j} \\ &= Ax_{l_i} + x_{l_j} + (y_{l_i} - y_{l_j}) + d_2(l_j, h_j) + y_{h_j}. \end{aligned}$$

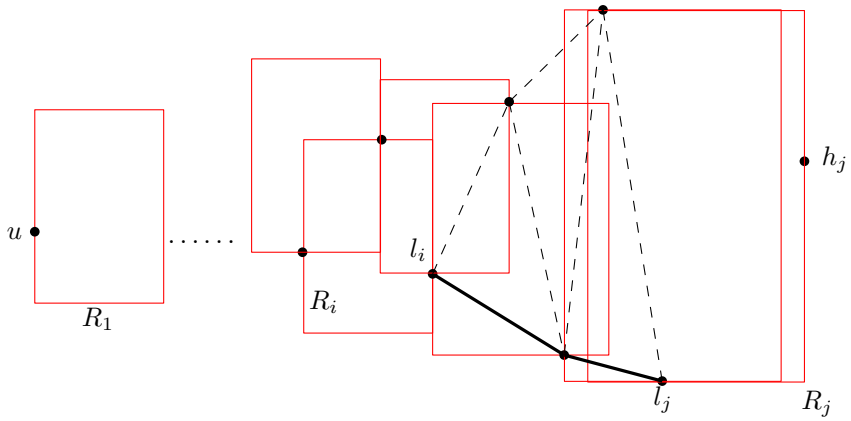



**Figure 3** In Property 2*a*, $R_j$ is the first inductive rectangle, $h_j$ is the inductive point of $R_j$, and $l_i, ..., l_{j-1} = l_j$ is the maximal low path ending at $l_j$. $R_i$ has potential and $l_i$ is on the E side of $R_i$.

Because $R_j$ is inductive, we know that edge $(l_j, h_j)$ is gentle. Therefore, $d_2(l_j, h_j) \leq \sqrt{1+A^2}(x_{h_j} - x_{l_j})$ and thus:

$$\begin{aligned} d_t(u, h_j) + (y_{h_j} - y) &\leq Ax_{l_i} + x_{l_j} + (y_{l_i} - y_{l_j}) + \sqrt{1+A^2}(x_{h_j} - x_{l_j}) + y_{h_j} \\ &\leq Ax_{l_i} + (y_{l_i} - y_{l_j}) + \sqrt{1+A^2}x_{h_j} + y_{h_j}. \end{aligned}$$

Furthermore, again because edge $(l_j, h_j)$ is gentle, we have that $y_{h_j} - y_{l_j} \leq A(x_{h_j} - x_{l_j})$ and therefore:

$$\begin{aligned} d_t(u, h_j) + (y_{h_j} - y) &\leq Ax_{l_i} + y_{l_i} + \sqrt{1+A^2}x_{h_j} + A(x_{h_j} - x_{l_j}) \\ &\leq y_{l_i} + \sqrt{1+A^2}x_{h_j} + Ax_{h_j}. \end{aligned}$$

Note that when $i \leq j$, then $x_{l_i} \leq x_{l_j}$. Finally, since $R(u, v)$ is empty, $l_i$ must lie below it and thus $y_{l_i} < 0$, which leads to: $d_t(u, h_j) + (y_{h_j} - y) \leq (A + \sqrt{A^2+1})x_{h_j}$.

*Property 2b:* Let $R_j$ be the first inductive rectangle in the sequence $R_1, ..., R_k$. Let $l_i, ..., l_{j-1} = l_j$ be the maximal low path ending at $l_j$, and recall that $h_j$ is the inductive point of $R_j$. By Lemma 4, $R_i$ has potential, and by Lemma 5, we have $d_t(u, l_i) \leq (1+L)x_{l_i}$. Since $L = \frac{1}{A}$, we have

$$\begin{aligned} d_t(u, h_j) + A(y_{h_j} - y) &\leq d_t(u, l_i) + d_t(l_i, l_j) + d_2(l_j, h_j) + A(y_{h_j} - y) \\ &\leq (1 + \tfrac{1}{A})x_{l_i} + d_t(l_i, l_j) + d_2(l_j, h_j) + Ay_{h_j}. \end{aligned}$$

Since $l_i, ..., l_{j-1} = l_j$ is a maximal low path, by Lemma 7 we know $d_t(l_i, l_j) \leq (x_{l_j} - x_{l_i}) + (y_{l_i} - y_{l_j})$. Because $R_j$ is inductive, we know that edge $(l_j, h_j)$ is gentle. Therefore, $d_2(l_j, h_j) \leq \sqrt{1 + \frac{1}{A^2}}(x_{h_j} - x_{l_j})$ and thus:

$$\begin{aligned} d_t(u, h_j) + A(y_{h_j} - y) &\leq (1 + \tfrac{1}{A})x_{l_i} + (x_{l_j} - x_{l_i}) + (y_{l_i} - y_{l_j}) + d_2(l_j, h_j) + Ay_{h_j} \\ &\leq (1 + \tfrac{1}{A})x_{l_i} + (x_{l_j} - x_{l_i}) + (y_{l_i} - y_{l_j}) + \sqrt{1 + \tfrac{1}{A^2}}(x_{h_j} - x_{l_j}) + Ay_{h_j} \\ &\leq \tfrac{1}{A}x_{l_i} + (y_{l_i} - y_{l_j}) + \sqrt{1 + \tfrac{1}{A^2}}x_{h_j} + Ay_{h_j}. \end{aligned}$$

Again because edge $(l_j, h_j)$ is gentle, we have that $y_{h_j} - y_{l_j} \leq \frac{1}{A}(x_{h_j} - x_{l_j})$. Therefore $A(y_{h_j} - y_{l_j}) \leq (x_{h_j} - x_{l_j})$. We have $A \geq 1$ and therefore:

$$\begin{aligned} d_t(u, h_j) + A(y_{h_j} - y) &\leq \tfrac{1}{A}x_{l_i} + A(y_{l_i} - y_{l_j}) + \sqrt{1 + \tfrac{1}{A^2}}x_{h_j} + Ay_{h_j} \\ &\leq \tfrac{1}{A}x_{l_i} + Ay_{l_i} + (x_{h_j} - x_{l_j}) + \sqrt{1 + \tfrac{1}{A^2}}x_{h_j}. \end{aligned}$$

Since $\frac{1}{A} \leq 1$ and $i \leq j$, we have $\frac{1}{A}x_{l_i} \leq x_{l_i} \leq x_{l_j}$. Therefore

$$d_t(u, h_j) + A(y_{h_j} - y) \leq Ay_{l_i} + x_{h_j} + \sqrt{1 + \tfrac{1}{A^2}}x_{h_j} \leq (1 + \sqrt{\tfrac{1}{A^2} + 1})x_{h_j}.$$

*Property 2c:* Let $R_j$ be the first inductive rectangle in the sequence $R_1, ..., R_k$. Now, let $h_i, ..., h_{j-1} = h_j$ be the maximal high path ending at $h_j$, and recall that $l_j$ is the inductive point of $R_j$. By Lemma 4, $R_i$ has potential, and by Lemma 5, we have $d_t(u, h_i) \leq (1 + L)x_{h_i}$. Since $L = A$,

$$\begin{aligned} d_t(u, l_j) - y_{l_j} &\leq d_t(u, h_i) + d_t(h_i, h_j) + d_2(h_j, l_j) - y_{l_j} \\ &\leq (1 + A)x_{h_i} + d_t(h_i, h_j) + d_2(h_j, l_j) - y_{l_j}. \end{aligned}$$

Since $h_i, ..., h_{j-1} = h_j$ is a maximal high path, by Lemma 7 we know $d_t(h_i, h_j) \leq (x_{h_j} - x_{h_i}) + (y_{h_j} - y_{h_i})$. It follows that:

$$\begin{aligned} d_t(u, l_j) - y_{l_j} &\leq (1 + A)x_{h_i} + (x_{h_j} - x_{h_i}) + (y_{h_j} - y_{h_i}) + d_2(h_j, l_j) - y_{l_j} \\ &= Ax_{h_i} + x_{h_j} + (y_{h_j} - y_{h_i}) + d_2(h_j, l_j) - y_{l_j}. \end{aligned}$$

Because $R_j$ is inductive, we know that edge $(l_j, h_j)$ is gentle. Therefore, $d_2(h_j, l_j) \leq \sqrt{1 + A^2}(x_{l_j} - x_{h_j})$ and thus:

$$\begin{aligned} d_t(u, l_j) - y_{l_j} &\leq Ax_{h_i} + x_{h_j} + (y_{h_j} - y_{h_i}) + \sqrt{1 + A^2}(x_{l_j} - x_{h_j}) - y_{l_j} \\ &\leq Ax_{h_i} + (y_{h_j} - y_{h_i}) + \sqrt{1 + A^2}x_{l_j} - y_{l_j}. \end{aligned}$$

Furthermore, again because edge $(l_j, h_j)$ is gentle, we have that $y_{h_j} - y_{l_j} \leq A(x_{l_j} - x_{h_j})$ and therefore:

$$\begin{aligned} d_t(u, l_j) - y_{l_j} &\leq Ax_{h_i} - y_{h_i} + \sqrt{1 + A^2}x_{l_j} + A(x_{l_j} - x_{h_j}) \\ &\leq -y_{h_i} + \sqrt{1 + A^2}x_{l_j} + Ax_{l_j}. \end{aligned}$$

Finally, since $R(u, v)$ is empty, $h_i$ must lie above it and thus $y_{h_i} > 0$, which leads to $d_t(u, l_j) - y_{l_j} \leq (A + \sqrt{A^2 + 1})x_{l_j}$.

*Property 2d:* Let $R_j$ be the first inductive rectangle in the sequence $R_1, ..., R_k$. Now, let $h_i, ..., h_{j-1} = h_j$ be the maximal high path ending at $h_j$, and recall that $l_j$ is the inductive point of $R_j$. By Lemma 4, $R_i$ has potential, and by Lemma 5, we have $d_t(u, h_i) \leq (1 + L)x_{h_i}$. Since $L = \frac{1}{A}$,

$$\begin{aligned} d_t(u, l_j) - Ay_{l_j} &\leq d_t(u, h_i) + d_t(h_i, h_j) + d_2(h_j, l_j) - Ay_{l_j} \\ &\leq (1 + \tfrac{1}{A})x_{h_i} + d_t(h_i, h_j) + d_2(h_j, l_j) - Ay_{l_j}. \end{aligned}$$

Since $h_i, ..., h_{j-1} = h_j$ is a maximal high path, by Lemma 7 we know $d_t(h_i, h_j) \leq (x_{h_j} - x_{h_i}) + (y_{h_j} - y_{h_i})$. Because edge $(l_j, h_j)$ is gentle, we have that $d_2(h_j, l_j) \leq \sqrt{1 + \frac{1}{A^2}}(x_{l_j} - x_{h_j})$. It follows that:

$$\begin{aligned} d_t(u,l_j) - Ay_{l_j} &\leq (1+\tfrac{1}{A})x_{h_i} + (x_{h_j} - x_{h_i}) + (y_{h_j} - y_{h_i}) + d_2(h_j,l_j) - Ay_{l_j} \\ &\leq (1+\tfrac{1}{A})x_{h_i} + (x_{h_j} - x_{h_i}) + (y_{h_j} - y_{h_i}) + \sqrt{1+\tfrac{1}{A^2}}(x_{l_j} - x_{h_j}) - Ay_{l_j} \\ &\leq \tfrac{1}{A}x_{h_i} + (y_{h_j} - y_{h_i}) + \sqrt{1+\tfrac{1}{A^2}}x_{l_j} - Ay_{l_j}. \end{aligned}$$

Again because edge $(l_j, h_j)$ is gentle, we have that $y_{h_j} - y_{l_j} \leq \frac{1}{A}(x_{l_j} - x_{h_j})$. Therefore $A(y_{h_j} - y_{l_j}) \leq (x_{l_j} - x_{h_j})$ and

$$\begin{aligned} d_t(u,l_j) - Ay_{l_j} &\leq \tfrac{1}{A}x_{h_i} + A(y_{h_j} - y_{h_i}) + \sqrt{1+\tfrac{1}{A^2}}x_{l_j} - Ay_{l_j} \\ &\leq \tfrac{1}{A}x_{h_i} + (x_{l_j} - x_{h_j}) - Ay_{h_i} + \sqrt{1+\tfrac{1}{A^2}}x_{l_j}. \end{aligned}$$

Since $\frac{1}{A} \leq 1$, we have $\frac{1}{A}x_{h_i} \leq x_{h_i} \leq x_{h_j}$. Thus

$$\begin{aligned} d_t(u,l_j) - Ay_{l_j} &\leq x_{l_j} - Ay_{h_i} + \sqrt{1+\tfrac{1}{A^2}}x_{l_j} \\ &\leq (1+\sqrt{\tfrac{1}{A^2}+1})x_{l_j}. \end{aligned}$$

as required, completing our proof of Property 1, 2*a*, 2*b*, 2*c* and 2*d*. ◀

Our final ingredient determines the types of edges we can encounter when the $y$-coordinate of a vertex differs significantly from that of $v$.

▶ **Lemma 9.** *Let $R(u,v)$ not contain any vertices of $P$ and let the coordinates of the inductive point $c$ of $R_i$ be such that it satisfies $0 < L(x - x_c) < |y - y_c|$.*

- *If $c = h_i$ and thus $0 < L(x - x_c) < y_c - y$, then let $j$ be the smallest index larger than $i$ such that $L(x - x_{h_j}) \geq y_c - y \geq 0$. All edges on the path $h_i, ..., h_j$ are NE edges.*
- *If $c = l_i$ and thus $0 < L(x - x_c) < y - y_c$, then let $j$ be the smallest index larger than $i$ such that $L(x - x_{l_j}) \geq y - y_c \geq 0$. All edges on the path $l_i, ..., l_j$ are SE edges.*

We now have all the ingredients needed to prove our main result. Recall that, up to Lemma 9, the $(x,y)$-coordinate system is fixed so that $Ld_x(u,v) > d_y(u,v)$, i.e. $Lx \geq y$. However, for ease of exposition, in Theorem 10 we instead fix the $(x,y)$-coordinate system so that all the homothet rectangles have their vertical sides being the long sides.

Note that in Lemma 8, we obtain different upper bounds depending on whether $L = A$ or $L = 1/A$. These two cases must be treated differently for the inductive proof of Theorem 10 to hold. In particular, in Theorem 10 the bound for $Ad_x(u,v) \geq d_y(u,v)$ does not coincide with the rotated version of the bound for $Ad_x(u,v) < d_y(u,v)$.

▶ **Theorem 10.** *Let $u, v$ be any two vertices in the rectangle Delaunay triangulation. If $Ad_x(u,v) \geq d_y(u,v)$, then*

$$d_t(u,v) \leq (A + \sqrt{A^2+1})x + y.$$

*Otherwise,*

$$d_t(u,v) \leq Ax + \left(1 + \sqrt{\tfrac{1}{A^2}+1}\right)y.$$

**Proof.** We consider all pairs of vertices $(u,v)$ and order them by the size of the smallest scaled translate of $R$ that has both $u$ and $v$ on its boundary. We perform induction based on the rank in this ordering.

The first pair $(u, v)$ in this ordering has the smallest overall scaled translate of $R$ and can thus contain no vertices of $P$, as any such vertex would imply the existence of a smaller rectangle with two vertices on its boundary, contradicting that we are considering the smallest one. Hence, by construction there exists an edge between $u$ and $v$ and thus $d_t(u, v) = d_2(u, v) \leq x + y$, satisfying the induction hypothesis, regardless of whether or not $Ad_x(u, v) \geq d_y(u, v)$.

Next, consider an arbitrary pair $(u, v)$ and assume the theorem holds for all pairs $(u, v)$ defining a smaller rectangle. We consider two cases: $R(u, v)$ does not contain any vertex of $P$, and $R(u, v)$ contains some vertices of $P$.

**Case 1:** There are no vertices inside $R(u, v)$. We distinguish two subcases, either $Ax \geq y$ or $Ax < y$.

**Subcase $Ax \geq y$:** Note that since the vertical side of the homothets is the longer side, for the $(x, y)$-coordinate system we have $L = A$, and $Lx \geq y$.

If $(u, v)$ is an edge in the rectangle Delaunay triangulation, then $d_t(u, v) \leq x + y \leq (A + \sqrt{A^2 + 1})x + y$. Otherwise, if no rectangle in $R_1, ..., R_k$ is inductive then by Property 1 of Lemma 8 we know $d_t(u, v) \leq (A + \sqrt{A^2 + 1})x + y$.

Hence, we focus on the case where there is an inductive rectangle. Let $R_i$ be the first inductive rectangle in the sequence $R_1, ..., R_k$. We distinguish the case where the inductive point is $h_i$ and where it is $l_i$. If $h_i$ is the inductive point of $R_i$ then by Property 2$a$ of Lemma 8 we know $d_t(u, h_i) + (y_{h_i} - y) \leq (A + \sqrt{A^2 + 1})x_{h_i}$ and thus $d_t(u, h_i) \leq (A + \sqrt{A^2 + 1})x_{h_i} - (y_{h_i} - y)$.

If $A(x - x_{h_i}) \geq y_{h_i} - y \geq 0$, we let $h_j = h_i$ in the remainder. Otherwise, we let $j$ be the smallest index larger than $i$ such that $A(x - x_{h_j}) \geq y_{h_j} - y \geq 0$. By Lemma 9, $h_j$ exists and all edges on the path $h_i, ..., h_j$ are NE edges. By triangle inequality, $d_t(h_m, h_{m+1}) \leq (x_{h_{m+1}} - x_{h_m}) + (y_{h_m} - y_{h_{m+1}})$ for any $h_m$ and $h_{m+1}$ on this path. This implies that $d_t(h_i, h_j) \leq (x_{h_j} - x_{h_i}) + (y_{h_i} - y_{h_j})$. Since $A(x - x_{h_j}) \geq y_{h_j} - y \geq 0$ and the smallest scaled translate of $R$ with $h_j$ and $v$ on its boundary is smaller than that of $u$ and $v$, we can use induction to get $d_t(h_j, v) \leq (A + \sqrt{A^2 + 1})d_x(h_j, v) + d_y(h_j, v)$. Putting everything together, we obtain that

$$\begin{aligned}
d_t(u, v) &\leq d_t(u, h_i) + d_t(h_i, h_j) + d_t(h_j, v) \\
&\leq (A + \sqrt{A^2 + 1})x_{h_i} - (y_{h_i} - y) + (x_{h_j} - x_{h_i}) + (y_{h_i} - y_{h_j}) \\
&\quad + (A + \sqrt{A^2 + 1})d_x(h_j, v) + d_y(h_j, v) \\
&= (A + \sqrt{A^2 + 1})x_{h_i} + (x_{h_j} - x_{h_i}) + (y - y_{h_j}) \\
&\quad + (A + \sqrt{A^2 + 1})d_x(h_j, v) + d_y(h_j, v) \\
&\leq (A + \sqrt{A^2 + 1})d_x(u, h_j) - d_y(h_j, v) + (A + \sqrt{A^2 + 1})d_x(h_j, v) + d_y(h_j, v) \\
&= (A + \sqrt{A^2 + 1})x.
\end{aligned}$$

proving the theorem when $h_i$ is the inductive point of $R_i$.

If $l_i$ is the inductive point of $R_i$ then by Property 2$c$ of Lemma 8 we know $d_t(u, l_i) - y_{l_i} \leq (A + \sqrt{A^2 + 1})x_{l_i}$ and thus $d_t(u, l_i) \leq (A + \sqrt{A^2 + 1})x_{l_i} + y_{l_i}$.

If $A(x - x_{l_i}) \geq y - y_{l_i}$, we let $l_j = l_i$ in the remainder. Otherwise, we let $j$ be the smallest index larger than $i$ such that $A(x - x_{l_j}) \geq y - y_c \geq 0$. By Lemma 9, $l_j$ exists and all edges on the path $l_i, ..., l_j$ are SE edges. By triangle inequality, $d_t(l_m, l_{m+1}) \leq (x_{l_{m+1}} - x_{l_m}) + (y_{l_{m+1}} - y_{h_m})$ for any $l_m$ and $l_{m+1}$ on this path. This implies that $d_t(l_i, l_j) \leq (x_{l_j} - x_{l_i}) + (y_{l_j} - y_{l_i})$. Since $A(x - x_{l_j}) \geq y - y_{l_j} \geq 0$ and the smallest scaled translate of $R$ with $l_j$ and $v$ on its boundary is smaller than that of $u$ and $v$, we can use induction to get $d_t(l_j, v) \leq (A + \sqrt{A^2 + 1})d_x(l_j, v) + d_y(l_j, v)$. Putting everything together, this implies that

$$\begin{aligned} d_t(u,v) &\leq d_t(u,l_i) + d_t(l_i,l_j) + d_t(l_j,v) \\ &\leq (A+\sqrt{A^2+1})x_{l_i} + y_{l_i} + (x_{l_j} - x_{l_i}) + (y_{l_j} - y_{l_i}) \\ &\quad + (A+\sqrt{A^2+1})d_x(l_j,v) + d_y(l_j,v) \\ &\leq (A+\sqrt{A^2+1})x_{l_i} + (x_{l_j} - x_{l_i}) + y_{l_j} + (A+\sqrt{A^2+1})d_x(l_j,v) + d_y(l_j,v) \\ &\leq (A+\sqrt{A^2+1})x + y. \end{aligned}$$

completing the proof of Case 1 when $Ax \geq y$.

**Subcase $Ax < y$:** Consider the $(x', y')$-coordinate system where the $x'$-axis equals the $y$-axis and the $y'$-axis equals the $x$-axis. See Figure 4. When we look at the homothet rectangles $R_1, \ldots, R_k$ intersecting the segment $uv$ in the $(x', y')$-coordinate system, the horizontal side of the homothets is the longer side and we have $L = 1/A$. Therefore, $Ax < y$ implies $Lx' > y'$.

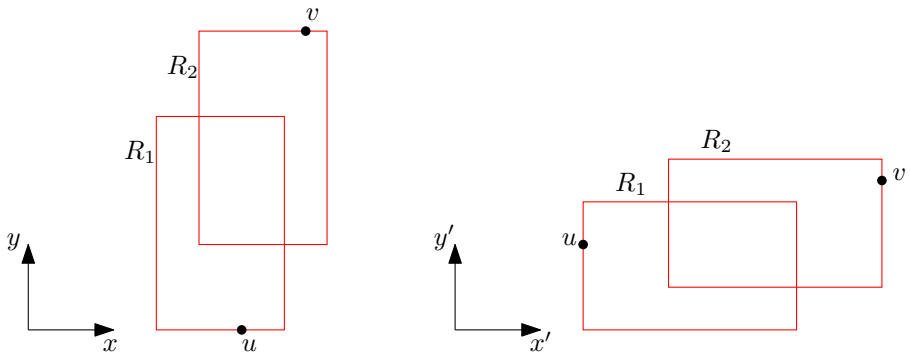



**Figure 4** Homothet rectangles $R_1, \ldots, R_k$ in the $(x', y')$-coordinate system, for $k = 2$.

If $(u, v)$ is an edge in the rectangle Delaunay triangulation, then $d_t(u,v) \leq x' + y' \leq (1+\sqrt{\frac{1}{A^2}+1})y + Ax$. If no rectangle in $R_1, ..., R_k$ is inductive then by Property 1 of Lemma 8 we know $d_t(u,v) \leq (\frac{1}{A}+\sqrt{\frac{1}{A^2}+1})x' + y' \leq (1+\sqrt{\frac{1}{A^2}+1})y + Ax$.

When there is an inductive rectangle, define $R_i$, $h_i$ and $l_i$ as above. If $h_i$ is the inductive point of $R_i$ then by Property 2$b$ of Lemma 8 we know $d_t(u,h_i) + A(y'_{h_i} - y') \leq (1+\sqrt{\frac{1}{A^2}+1})x'_{h_i}$.

If $\frac{1}{A}(x' - x'_{h_i}) \geq y'_{h_i} - y' \geq 0$, we let $h_j = h_i$ in the remainder. Otherwise, we let $j$ be the smallest index larger than $i$ such that $\frac{1}{A}(x' - x'_{h_j}) \geq y'_{h_j} - y' \geq 0$. By Lemma 9, $h_j$ exists and all edges on the path $h_i, ..., h_j$ are NE edges. By triangle inequality, $d_t(h_i,h_j) \leq (x'_{h_j} - x'_{h_i}) + (y'_{h_i} - y'_{h_j})$. Since $\frac{1}{A}(x' - x'_{h_j}) \geq y'_{h_j} - y' \geq 0$ and the smallest scaled translate of $R$ with $h_j$ and $v$ on its boundary is smaller than that of $u$ and $v$, we can use induction to get $d_t(h_j,v) \leq (1+\sqrt{\frac{1}{A^2}+1})d_y(h_j,v) + Ad_x(h_j,v)$. Putting everything together, we obtain

$$\begin{aligned}
d_t(u,v) &\leq d_t(u,h_i) + d_t(h_i,h_j) + d_t(h_j,v) \\
&\leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{h_i} - A(y'_{h_i} - y') + (x'_{h_j} - x'_{h_i}) + (y'_{h_i} - y'_{h_j}) \\
&\quad + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(h_j,v) + Ad_x(h_j,v) \\
&\leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{h_i} - A(y'_{h_i} - y') + (x'_{h_j} - x'_{h_i}) + A(y'_{h_i} - y'_{h_j}) \\
&\quad + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(h_j,v) + Ad_x(h_j,v) \\
&\leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{h_i} + (x'_{h_j} - x'_{h_i}) - Ad_{y'}(h_j,v) \\
&\quad + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(h_j,v) + Ad_x(h_j,v).
\end{aligned}$$

Recall that the $y'$-axis in the $(x',y')$-coordinate system equals the $x$-axis in the $(x,y)$-coordinate system, so $Ad_{y'}(h_j,v) = Ad_x(h_j,v)$. Thus

$$\begin{aligned}
d_t(u,v) &\leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{h_j} + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(h_j,v) \\
&= (1+\sqrt{\tfrac{1}{A^2}+1})y.
\end{aligned}$$

If $l_i$ is the inductive point of $R_i$ then by Property 2$d$ of Lemma 8 we know $d_t(u,l_i) - Ay'_{l_i} \leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{l_i}$. Thus $d_t(u,l_i) \leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{l_i} + Ay'_{l_i}$.
If $\frac{1}{A}(x' - x'_{l_i}) \geq y' - y'_{l_i} \geq 0$, we let $l_j = l_i$ in the remainder. Otherwise, we let $j$ be the smallest index larger than $i$ such that $\frac{1}{A}(x' - x'_{l_j}) \geq y' - y'_{l_j} \geq 0$. By Lemma 9, $l_j$ exists and all edges on the path $l_i, ..., l_j$ are SE edges. By triangle inequality, $d_t(l_i,l_j) \leq (x'_{l_j} - x'_{l_i}) + (y'_{l_j} - y'_{l_i})$. Since $\frac{1}{A}(x' - x'_{l_j}) \geq y' - y'_{l_j} \geq 0$ and the smallest scaled translate of $R$ with $l_j$ and $v$ on its boundary is smaller than that of $u$ and $v$, we can use induction to get $d_t(l_j,v) \leq (1+\sqrt{\tfrac{1}{A^2}+1})d_y(l_j,v) + Ad_x(l_j,v)$. Thus we obtain that

$$\begin{aligned}
d_t(u,v) &\leq d_t(u,l_i) + d_t(l_i,l_j) + d_t(l_j,v) \\
&\leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{l_i} + Ay'_{l_i} + (x'_{l_j} - x'_{l_i}) + (y'_{l_j} - y'_{l_i}) \\
&\quad + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(l_j,v) + Ad_x(l_j,v) \\
&\leq (1+\sqrt{\tfrac{1}{A^2}+1})x'_{l_i} + Ay'_{l_i} + (x'_{l_j} - x'_{l_i}) + A(y'_{l_j} - y'_{l_i}) \\
&\quad + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(l_j,v) + Ad_x(l_j,v) \\
&= (1+\sqrt{\tfrac{1}{A^2}+1})x'_{l_i} + (x'_{l_j} - x'_{l_i}) + Ay'_{l_j} \\
&\quad + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(l_j,v) + Ad_x(l_j,v).
\end{aligned}$$

Using that $Ay'_{l_j} = Ax_{l_j}$, we obtain

$$\begin{aligned}
d_t(u,v) &\leq (1+\sqrt{\tfrac{1}{A^2}+1})y_{l_j} + (1+\sqrt{\tfrac{1}{A^2}+1})d_y(l_j,v) + Ax \\
&= (1+\sqrt{\tfrac{1}{A^2}+1})y + Ax.
\end{aligned}$$

completing the proof of Case 1.

**Case 2:** There are vertices of $P$ inside $R(u,v)$. We distinguish two subcases, either $Ax \geq y$ or $Ax < y$.

**Subcase $Ax \geq y$:** We split $R(u,v)$ into three regions formally defined as follows: $A = \{p \mid p \text{ is inside } R(u,v) \text{ such that } Ad_x(u,p) < d_y(u,p)\}$, $B = \{p \mid p \text{ is inside } R(u,v) \text{ such that } Ad_x(u,p) \geq d_y(u,p) \text{ and } Ad_x(p,v) \geq d_y(p,v)\}$, $C = \{p \mid p \text{ is inside } R(u,v) \text{ such that } Ad_x(p,v) < d_y(p,v)\}$. Informally, these three regions can be constructed by considering the line through $u$ and the line through $v$ parallel to the line through the diagonal of $R$ and labelling the resulting regions $A$, $B$, and $C$ from left to right (see Figure 5(a)).

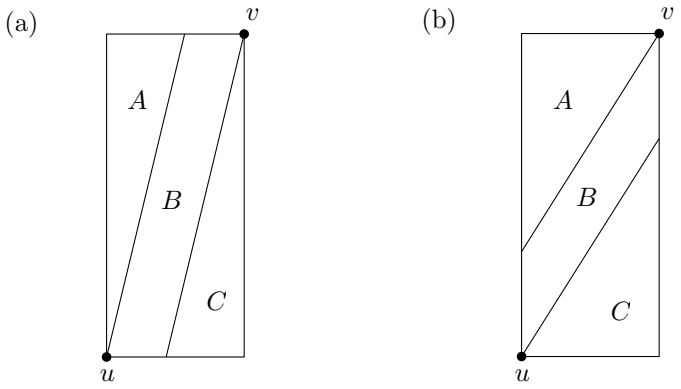



**Figure 5** (a) The three regions in $R(u,v)$ when $Ax \geq y$. (b) The three regions in $R(u,v)$ when $Ax < y$.

If there exists a vertex $p$ inside region $B$, then we can apply induction on the pairs $(u,p)$, which satisfies $Ad_x(u,p) \geq d_y(u,p)$, and $(p,v)$, which satisfies $Ad_x(p,v) \geq d_y(p,v)$:

$$\begin{aligned} d_t(u,v) &\leq d_t(u,p) + d_t(p,v) \\ &\leq (A + \sqrt{A^2+1})d_x(u,p) + d_y(u,p) + (A + \sqrt{A^2+1})d_x(p,v) + d_y(p,v) \\ &= (A + \sqrt{A^2+1})x + y. \end{aligned}$$

If there is no vertex inside region $B$, we define $R_u$ to be the smallest scaled translate of $R$ that has $u$ on its lower left corner and some vertex $p \in A$ in $R(u,v)$ on its boundary. Similarly, we define $R_v$ to be the smallest scaled translate of $R$ that has $v$ on its upper right corner and some vertex $q \in C$ in $R(u,v)$ on its boundary. Since $R(u,v)$ is not empty, at least one of $p$ and $q$ must exist. Assume without loss of generality that $p$ exists. In this case we have that $Ad_x(p,v) > d_y(p,v)$ and the smallest homothet with $p$ and $v$ on its boundary is smaller than that of $u$ and $v$. If $(u,p)$ is an edge in the rectangle Delaunay triangulation, then we obtain that:

$$\begin{aligned} d_t(u,v) &\leq d_t(u,p) + d_t(p,v) \\ &= d_2(u,p) + d_t(p,v) \\ &\leq d_x(u,p) + d_y(u,p) + (A + \sqrt{A^2+1})d_x(p,v) + d_y(p,v) \\ &\leq (A + \sqrt{A^2+1})x + y. \end{aligned}$$

An analogous argument can be used if $q$ exists and $(v,q)$ is an edge in the rectangle Delaunay triangulation.

Hence, it remains to consider the case where $(u, p)$ is not an edge, in which case $R_u$ is not empty. This implies that there exists a $p' \in C$ such that $(u, p')$ is an edge. We have that $Ad_x(p', v) < d_y(p', v)$ and the smallest scaled translate of $R$ with $p'$ and $v$ on its boundary is smaller than that of $u$ and $v$. By the induction hypothesis, we have:

$$\begin{aligned} d_t(u, v) &\leq d_t(u, p') + d_t(p', v) \\ &= d_2(u, p') + d_t(p', v) \\ &\leq d_x(u, p') + d_y(u, p') + Ad_x(p', v) + \left(1 + \sqrt{\tfrac{1}{A^2} + 1}\right) d_y(p', v) \\ &\leq Ax + \left(1 + \sqrt{\tfrac{1}{A^2} + 1}\right) y. \end{aligned}$$

Since $Ax \geq y$ and $\left(1 + \sqrt{\frac{1}{A^2} + 1}\right) > 1$, we have

$$\begin{aligned} d_t(u, v) &\leq 1Ax + \left(1 + \sqrt{\tfrac{1}{A^2} + 1}\right) y \\ &\leq \left(1 + \sqrt{\tfrac{1}{A^2} + 1}\right) Ax + 1y \\ &= \left(A + \sqrt{A^2 + 1}\right) x + y. \end{aligned}$$

**Subcase $Ax < y$:** We split $R(u, v)$ into three regions formally defined as follows: $A = \{p \mid p \text{ is inside } R(u, v) \text{ such that } Ad_x(v, p) \geq d_y(v, p)\}$, $B = \{p \mid p \text{ is inside } R(u, v) \text{ such that } Ad_x(v, p) < d_y(v, p) \text{ and } Ad_x(u, p) < d_y(u, p)\}$, $C = \{p \mid p \text{ is inside } R(u, v) \text{ such that } Ad_x(u, p) \geq d_y(u, p)\}$. See Figure 5(b).
If there exists a vertex $p$ inside region $B$, then we can apply induction on the pairs $(u, p)$, which satisfies $Ad_x(u, p) < d_y(u, p)$, and $(p, v)$, which satisfies $Ad_x(v, p) < d_y(v, p)$:

$$\begin{aligned} d_t(u, v) &\leq d_t(u, p) + d_t(p, v) \\ &\leq Ad_x(u, p) + (1 + \sqrt{\tfrac{1}{A^2} + 1}) d_y(u, p) + Ad_x(p, v) + (1 + \sqrt{\tfrac{1}{A^2} + 1}) d_y(p, v) \\ &= Ax + (1 + \sqrt{\tfrac{1}{A^2} + 1}) y. \end{aligned}$$

If there is no vertex inside region $B$, we define $R_u$ to be the smallest scaled translate of $R$ that has $u$ on its lower left corner and some vertex $p \in A$ in $R(u, v)$ on its boundary. Similarly, we define $R_v$ to be the smallest scaled translate of $R$ that has $v$ on its upper right corner and some vertex $q \in C$ in $R(u, v)$ on its boundary. Since $R(u, v)$ is not empty, at least one of $p$ and $q$ must exist. Assume without loss of generality that $p$ exists. In this case we have that $Ad_x(p, v) > d_y(p, v)$ and the smallest rectangle with $p$ and $v$ on its boundary is smaller than that of $u$ and $v$. If $(u, p)$ is an edge in the rectangle Delaunay triangulation, then we obtain that:

$$\begin{aligned} d_t(u, v) &\leq d_t(u, p) + d_t(p, v) \\ &= d_2(u, p) + d_t(p, v) \\ &\leq d_x(u, p) + d_y(u, p) + (A + \sqrt{A^2 + 1}) d_x(p, v) + d_y(p, v) \\ &\leq (A + \sqrt{A^2 + 1}) x + y. \end{aligned}$$

Since $Ax < y$, we have

$$\begin{aligned} d_t(u,v) &\leq (A+\sqrt{A^2+1})x + y \\ &= (1+\sqrt{1+\tfrac{1}{A^2}})Ax + 1y \\ &\leq Ax + (1+\sqrt{\tfrac{1}{A^2}+1})y. \end{aligned}$$

An analogous argument can be used if $q$ exists and $(v,q)$ is an edge in the rectangle Delaunay triangulation.
Hence, it remains to consider the case where $(u,p)$ is not an edge, in which case $R_u$ is not empty. This implies that there exists a $p' \in C$ such that $(u,p')$ is an edge. We have that $Ad_x(p',v) < d_y(p',v)$ and the smallest scaled translate of $R$ with $p'$ and $v$ on its boundary is smaller than that of $u$ and $v$. By the induction hypothesis, we have:

$$\begin{aligned} d_t(u,v) &\leq d_t(u,p') + d_t(p',v) \\ &= d_2(u,p') + d_t(p',v) \\ &\leq d_x(u,p') + d_y(u,p') + Ad_x(p',v) + \left(1+\sqrt{\tfrac{1}{A^2}+1}\right) d_y(p',v) \\ &\leq Ax + \left(1+\sqrt{\tfrac{1}{A^2}+1}\right) y. \end{aligned}$$

This completes the proof of Case 2 and the theorem. ◀

We can now use Theorem 10 to show an upper bound of the spanning ratio of the rectangle Delaunay triangulation. For any pair of vertices $u, v$ in the graph, if $Ad_x(u,v) \geq d_y(u,v)$ we have

$$\frac{d_t(u,v)}{d_2(u,v)} < \frac{(A+\sqrt{A^2+1})x + y}{\sqrt{x^2+y^2}}.$$

This function is maximized when $y/x = 1/(A+\sqrt{A^2+1})$, where the function is equal to

$$\sqrt{2}\sqrt{1+A^2+A\sqrt{1+A^2}}.$$

On the other hand, when $Ad_x(u,v) < d_y(u,v)$, we can get

$$\frac{d_t(u,v)}{d_2(u,v)} < \frac{Ax + (1+\sqrt{\frac{1}{A^2}+1})y}{\sqrt{x^2+y^2}}.$$

This function is maximized when $y/x = (1+\sqrt{\frac{1}{A^2}+1})/A$, where the function value equals

$$\sqrt{A^2+2+2\sqrt{1+\tfrac{1}{A^2}}+\tfrac{1}{A^2}},$$

which is at most $\sqrt{2}\sqrt{1+A^2+A\sqrt{1+A^2}}$. This implies the main result of the paper.

▶ **Theorem 11.** *The spanning ratio of the rectangle Delaunay triangulation is at most* $\sqrt{2}\sqrt{1+A^2+A\sqrt{1+A^2}}$*, where A is the aspect ratio of the rectangle used in its construction.*

Since it was already known that $\sqrt{2}\sqrt{1+A^2+A\sqrt{1+A^2}}$ is a lower bound on the spanning ratio [3], we obtain that the bound of $\sqrt{2}\sqrt{1+A^2+A\sqrt{1+A^2}}$ is tight.

## References


1. Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perković. Tight stretch factors for $L_1$- and $L_\infty$-Delaunay triangulations. *Computational Geometry: Theory and Applications (CGTA)*, 48(3):237–250, 2015.
2. Prosenjit Bose, Paz Carmi, Sébastien Collette, and Michiel Smid. On the stretch factor of convex Delaunay graphs. *Journal of Computational Geometry (JoCG)*, 1(1):41–56, 2010.
3. Prosenjit Bose, Jean-Lou De Carufel, and André van Renssen. Constrained generalized Delaunay graphs are plane spanners. *Computational Geometry: Theory and Applications (CGTA)*, 74:50–65, 2018.
4. Prosenjit Bose, Luc Devroye, Maarten Löffler, Jack Snoeyink, and Vishal Verma. Almost all Delaunay triangulations have stretch factor greater than $\pi/2$. *Computational Geometry: Theory and Applications (CGTA)*, 44(2):121–127, 2011.
5. Prosenjit Bose and Michiel Smid. On plane geometric spanners: A survey and open problems. *Computational Geometry: Theory and Applications (CGTA)*, 46(7):818–830, 2013.
6. L. Paul Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG)*, pages 169–177, 1986.
7. L. Paul Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences (JCSS)*, 39(2):205–219, 1989.
8. David P. Dobkin, Steven J. Friedman, and Kenneth J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry (DCG)*, 5(1):399–407, 1990.
9. J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry (DCG)*, 7(1):13–28, 1992.
10. Der-Tsai Lee and Arthur K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete & Computational Geometry (DCG)*, 1(3):201–217, 1986.
11. Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.
12. Ljubomir Perković, Michael Dennis, and Duru Türkoğlu. The stretch factor of hexagon-Delaunay triangulations. *Journal of Computational Geometry (JoCG)*, 12(2):86–125, 2021.
13. André van Renssen, Yuan Sha, Yucheng Sun, and Sampson Wong. The tight spanning ratio of the rectangle Delaunay triangulation, 2022. `arXiv:2211.11987`.
14. Ge Xia. The stretch factor of the Delaunay triangulation is less than 1.998. *SIAM Journal on Computing (SICOMP)*, 42(4):1620–1659, 2013.
15. Ge Xia and Liang Zhang. Toward the tight bound of the stretch factor of Delaunay triangulations. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG 2011)*, pages 175–180, 2011.

# Canonization of a Random Graph by Two Matrix-Vector Multiplications

**Oleg Verbitsky**
Institut für Informatik, Humboldt-Universität zu Berlin, Germany

**Maksim Zhukovskii**
Department of Computer Science, University of Sheffield, UK

## Abstract

We show that a canonical labeling of a random $n$-vertex graph can be obtained by assigning to each vertex $x$ the triple $(w_1(x), w_2(x), w_3(x))$, where $w_k(x)$ is the number of walks of length $k$ starting from $x$. This takes time $\mathcal{O}(n^2)$, where $n^2$ is the input size, by using just two matrix-vector multiplications. The linear-time canonization of a random graph is the classical result of Babai, Erdős, and Selkow. For this purpose they use the well-known combinatorial color refinement procedure, and we make a comparative analysis of the two algorithmic approaches.

**2012 ACM Subject Classification** Mathematics of computing → Random graphs; Mathematics of computing → Graph algorithms

**Keywords and phrases** Graph Isomorphism, canonical labeling, random graphs, walk matrix, color refinement, linear time

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.100

**Funding** *Oleg Verbitsky*: Supported by DFG grant KO 1053/8–2. On leave from the IAPMM, Lviv, Ukraine.

## 1 Introduction

A *walk* in a graph $G = (V, E)$ is a sequence of vertices $x_0x_1 \dots x_k$ such that $(x_i, x_{i+1}) \in E$ for every $0 \le i < k$. We say that $x_0x_1 \dots x_k$ is a walk of *length* $k$ *from* $x_0$ *to* $x_k$. For a vertex $x \in V$, let $w_k^G(x)$ denote the total number of walks of length $k$ in $G$ starting from $x$. Furthermore, we define $\mathbf{w}_k^G(x) = (w_1^G(x), \dots, w_k^G(x))$.

The Erdős-Rényi random graph $G(n, p)$ is a graph on the vertex set $[n] = \{1, \dots, n\}$ where each pair of distinct vertices $x$ and $y$ is adjacent with probability $p$ independently of the other pairs. In particular, $G(n, 1/2)$ is a random graph chosen equiprobably from among all graphs on $[n]$.

▶ **Theorem 1.** *Let $G = G(n, 1/2)$. Then*

$$\mathbf{w}_3^G(x) \neq \mathbf{w}_3^G(y) \text{ for all } x \neq y$$

*with probability at least $1 - O(\sqrt[4]{\ln n / n})$.*

If $\alpha$ is an isomorphism from a graph $G$ to a graph $H$, then clearly $\mathbf{w}_k^G(x) = \mathbf{w}_k^H(\alpha(x))$. Theorem 1, therefore, shows that the map $x \mapsto \mathbf{w}_3^G(x)$ is a canonical labeling of $G$ for almost all $n$-vertex graphs $G$. This labeling is easy to compute. Indeed, if $A$ is the adjacency matrix of $G$ and $\mathbb{1}$ is the all-ones vector-column of length $n$, then

$$(w_k^G(1), \dots, w_k^G(n))^\top = A^k \mathbb{1}.$$

After noting that $w_1^G(x) = d(x)$, where $d(x)$ denotes the degree of a vertex $x$, this yields the following simple canonical labeling algorithm.


**Algorithm A.** Canonical labeling of a random graph.

INPUT: a graph $G$ on $[n]$ with adjacency matrix $A$.

1. Form a vector $D_1 = (d(1), \dots, d(n))^\top$.
2. Compute $D_2 = AD_1$ and $D_3 = AD_2$.
3. Let $W$ be the matrix formed by the three columns $D_1, D_2, D_3$ and let $W_1, \dots, W_n$ be the rows of $W$.
4. If there are identical rows $W_x = W_y$ for some $x \neq y$, then give up. Otherwise,
5. to each vertex $x$, assign the label $W_x$.

▶ **Corollary 2.** *Algorithm A with high probability canonizes a random $n$-vertex graph, taking time $\mathcal{O}(n^2)$ on every input.*

The notation $\mathcal{O}(\cdot)$ in the time bound means a linear function up to the logarithmic factor $\log n \log\log n$ corresponding to the complexity of integer multiplication [7], that is, $\mathcal{O}(n^2) = O(n^2 \log n \log\log n)$. If the model of computation assumes that multiplication of two integers takes a constant time, then we just set $\mathcal{O}(n^2) = O(n^2)$. This time bound stems from the fact that the two matrix-vector multiplications in Step 2 are the most expensive operations performed by the algorithm. Note that this bound is essentially linear because a random graph is with high probability dense, having $(1/4 + o(1))n^2$ edges.

The linear-time canonization of almost all graphs is a classical result of Babai, Erdős, and Selkow [2], which was a basis for settling the average-case complexity of graph isomorphism in [3]. While our algorithm is based solely on basic linear-algebraic primitives, the method used in [2, 3] is purely combinatorial. Before comparing the two approaches, we put Theorem 1 in the context of the earlier work on walk counts and their applications to isomorphism testing.

Of course, Algorithm A can be enhanced by taking into account also longer walks, that is, by involving also other vector-columns $A^k\mathbb{1}$ for $k > 3$. Note that there is no gain in considering these vectors for $k \geq n$. Indeed, if $A^k\mathbb{1}$ is a linear combination of the vectors $\mathbb{1}, A\mathbb{1}, A^2\mathbb{1}, \dots, A^{k-1}\mathbb{1}$, the same is obviously true also for $A^{k+1}\mathbb{1}$ (cf. [11, Lemma 1] and see also [6] for a more detailed linear-algebraic analysis). Therefore, it suffices to start our matrix $W$ from the column $\mathbb{1}$ and add a subsequent column $A^k\mathbb{1}$ as long as this increases the rank of $W$, which is possible only up to $k = n - 1$. The $n \times n$ matrix $W = W^G$ formed by the columns $\mathbb{1}, A\mathbb{1}, \dots, A^{n-1}\mathbb{1}$ is called the *walk matrix* of the graph $G$ (*WM* for brevity). The entries of $W^G = (w_{x,k})_{1\leq x\leq n,\, 0\leq k<n}$ are nothing else as the walk counts $w_{x,k} = w_k^G(x)$. Note that $w_0^G(x) = 1$ as there is a single walk of length 0 from $x$.

We say that a graph $G$ is *WM-discrete* if the rows of the walk matrix $W^G$ are pairwise different, i.e., $W_x^G \neq W_y^G$ for all $x \neq y$. For a such $G$, the walk matrix yields a canonical labeling where each vertex $x$ is assigned the vector $W_x^G = (w_0^G(x), w_1^G(x), \dots, w_{n-1}^G(x))$. Note that if $W^G$ has identical rows, then this matrix is singular; cf. [5, Section 7]. O'Rourke and Touri [10] prove that the walk matrix of a random graph is non-singular with high probability. This implies that a random graph is WM-discrete with high probability and, hence, almost all graphs are canonizable by computing the $n \times n$ walk matrix similarly to Algorithm A. Note that this takes time $O(n^3)$, which is outperformed by our Corollary 2 due to using the truncated variant of WM of size $n \times 4$.

Remarkably, non-singular walk matrices can be used to test isomorphism of two given graphs directly rather than by computing their canonical forms. If graphs $G$ and $H$ are isomorphic, then their walk matrices $W^G$ and $W^H$ can be obtained from one another by

rearranging the rows. If the last condition is satisfied, we write $W^G \approx W^H$. This relation between matrices is efficiently checkable just by sorting the rows in the lexicographic order. We say that a graph $G$ is *WM-identifiable* if, conversely, for all $H$ we have $G \cong H$ whenever $W^G \approx W^H$. Liu and Siemons [8] prove that if the walk matrix of a graph is non-singular, then it uniquely determines the adjacency matrix. This implies by [10] that a random graph is WM-identifiable with high probability.

Note that, by a simple counting argument, almost all $n$-vertex graphs *cannot* be identified by the shorter version of the walk matrix of size $n \times k$ as long as $k = o(\sqrt{n/\log n})$. In particular, Theorem 1 cannot be extended to the identifiability concept.

The combinatorial approach of Babai, Erdős, and Selkow [2] is based on the *color refinement* procedure (*CR* for brevity) dating back to the sixties (e.g., [9]). CR begins with a uniform coloring of all vertices in an input graph and iteratively refines a current coloring according to the following principle: If two vertices are equally colored but have distinct color frequencies in their neighborhoods, then they get distinct colors in the next refinement step. The refinement steps are executed as long as the refinement is proper. As soon as the color classes stay the same, CR terminates and outputs the current coloring (a detailed description of the algorithm is given in Section 3.1). CR *distinguishes* graphs $G$ and $H$ if their color palettes are distinct. A graph $G$ is called *CR-identifiable* if it is distinguishable by CR from every non-isomorphic $H$. CR can also be used for computing a canonical labeling of a single input graph. We say that a graph $G$ is *CR-discrete* if CR assigns a unique color to each vertex of $G$. It is easy to prove that every CR-discrete graph is CR-identifiable. We do not know whether or not this is true also for the corresponding WM concepts.

Powers and Sulaiman [11] discuss examples when the CR-partition and the WM-partition are different, that is, CR and the WM-based vertex-classification algorithm give different results. In particular, [11, Fig. 3] shows a graph which is, in our terminology, CR-discrete but not WM-discrete. We give a finer information about the relationship between the two algorithmic approaches.

▶ **Theorem 3.**

1. *Every WM-discrete graph is also CR-discrete.*
2. *Every WM-identifiable graph is also CR-identifiable.*
3. *There is a graph that is*
   - **(a)** *CR-discrete (hence also CR-identifiable) and*
   - **(b)** *neither WM-discrete*
   - **(c)** *nor WM-identifiable.*

Theorem 3 shows that the WM approach is superseded by the CR algorithm with regard to canonization of a single input graph and testing isomorphism of two input graphs. Moreover, CR is sometimes more successful with respect to both algorithmic problems. Thus, WM can be regarded as a weaker algorithmic tool for canonical labeling and isomorphism testing, which is not so surprising as this approach is actually based on a single basic linear-algebraic primitive, namely matrix-vector multiplication. In this sense, Algorithm A is arguably simpler than the classical CR-based canonization of a random graph as it demonstrates that a random graph can be canonized in an essentially linear time even with less powerful computational means.

Theorems 1 and 3 are proved in Sections 2 and 3 respectively.

# 2 Canonization of a random graph

## 2.1 Probability preliminaries

Let $X$ be a binomial random variable with parameters $n$ and $p$, that is, $X = \sum_{i=1}^{n} X_i$ where $X_i$'s are mutually independent and, for each $i$, we have $X_i = 1$ with probability 1 and $X_i = 0$ with probability $1 - p$. We use the notation $X \sim \mathrm{Bin}(n, p)$ when $X$ has this distribution. As well known, $X$ is well-concentrated around its expectation $np$.

▶ **Lemma 4** (Chernoff's bound; see, e.g., [1, Corollary A.1.7]). *If $X \sim \mathrm{Bin}(n, p)$, then*

$$\mathsf{P}[|X - np| > t] \le 2e^{-2t^2/n}$$

*for every $t \ge 0$.*

▶ **Lemma 5.** *If $X$ and $Y$ are independent random variables, each having the probability distribution $\mathrm{Bin}(n, 1/2)$, then $\mathsf{P}[X = Y] < 1/\sqrt{\pi n}$.*

**Proof.** Using the well-known estimate

$$\binom{2n}{n} < \frac{2^{2n}}{\sqrt{\pi n}}, \tag{1}$$

we obtain

$$\mathsf{P}[X = Y] = \sum_{k=0}^{n} \left( \binom{n}{k} 2^{-n} \right)^2 = 2^{-2n} \sum_{k=0}^{n} \binom{n}{k}^2 = 2^{-2n} \binom{2n}{n} < \frac{1}{\sqrt{\pi n}},$$

where the last equality is a special case of Vandermonde's convolution. ◀

## 2.2 Proof of Theorem 1

For a vertex $i \in [n]$, recall that $\mathbf{w}_3(i) = (w_1^G(i), w_2^G(i), w_3^G(i))$. By the union bound,

$$\mathsf{P}[\mathbf{w}_3(i) = \mathbf{w}_3(j) \text{ for some } i, j] \le \sum_{i,j} \mathsf{P}[\mathbf{w}_3(i) = \mathbf{w}_3(j)] = \binom{n}{2} \mathsf{P}[\mathbf{w}_3(1) = \mathbf{w}_3(2)].$$

Therefore, it suffices to prove that

$$\mathsf{P}[\mathbf{w}_3(1) = \mathbf{w}_3(2)] = O(n^{-9/4} \ln^{1/4} n). \tag{2}$$

Let $N_H(v)$ denote the neighborhood of a vertex $v$ in a graph $H$. Given two sets $U_1 \subset [n] \setminus \{1\}$ and $U_2 \subset [n] \setminus \{2\}$, let $G' = G'(U_1, U_2)$ be the random graph $G$ subject to the conditions $N_G(1) = U_1$ and $N_G(2) = U_2$. In other terms, $G'$ is a random graph on $[n]$ chosen equiprobably among all graphs satisfying these conditions. Let $w'_k(i) = w_k^{G'}(i)$ denote the number of walks of length $k$ emanating from $i$ in $G'$ (the dependence of $w'_k(i)$ on the pair $U_1, U_2$ will be dropped for the sake of notational convenience). Define

$$p(U_1, U_2) = \mathsf{P}\left[ \sum_{i \in U_1} w'_1(i) = \sum_{i \in U_2} w'_1(i) \text{ and } \sum_{i \in U_1} w'_2(i) = \sum_{i \in U_2} w'_2(i) \right].$$

We have

$$\begin{aligned}&\mathsf{P}[\mathbf{w}_3(1)=\mathbf{w}_3(2)]\\&=\sum_{U_1,U_2\,:\,|U_1|=|U_2|}\mathsf{P}[\mathbf{w}_3(1)=\mathbf{w}_3(2)\mid N_G(1)=U_1,\ N_G(2)=U_2]\\&\qquad\times\mathsf{P}[N_G(1)=U_1,\,N_G(2)=U_2]\\&\qquad=\sum_{U_1,U_2\,:\,|U_1|=|U_2|}p(U_1,U_2)\times\mathsf{P}[N_G(1)=U_1,\,N_G(2)=U_2].\quad(3)\end{aligned}$$

Note first that

$$\begin{aligned}\sum_{|U_1|=|U_2|}\mathsf{P}\left[N_G(1)=U_1,\,N_G(2)=U_2\right]&=\mathsf{P}\left[|N_G(1)|=|N_G(2)|\right]\\&=\mathsf{P}\left[|N_G(1)\setminus\{2\}|=|N_G(2)\setminus\{1\}|\right]=O(n^{-1/2})\end{aligned}$$

by Lemma 5 because $|N_G(1)\setminus\{2\}|\sim\mathrm{Bin}(n-2,1/2)$ and $|N_G(2)\setminus\{1\}|\sim\mathrm{Bin}(n-2,1/2)$ are independent binomial random variables. This allows us to derive (2) from (3) if we prove that

$$p(U_1,U_2)=O(n^{-7/4}\ln^{1/4}n)\qquad(4)$$

for the neighborhood sets $U_1$ and $U_2$.

In fact, we do not need to prove (4) for all pairs $U_1,U_2$ because the contribution of some of them in (3) is negligible. Indeed, set $\varepsilon(n)=n^{-1/4}$. Note that $|N_G(j)|\sim\mathrm{Bin}(n-1,1/2)$ for $j=1,2$ and $|(N_G(1)\cap N_G(2))\setminus\{1,2\}|\sim\mathrm{Bin}(n-2,1/4)$. By the Chernoff bound (see Lemma 4), we have $(1/2-\varepsilon(n))n\le|N_G(j)|\le(1/2+\varepsilon(n))n$ for $j=1,2$ and $(1/4-\varepsilon(n))n\le|N_G(1)\cap N_G(2)|\le(1/4+\varepsilon(n))n$ with probability $1-e^{-\Omega(\sqrt{n})}$. Call a pair $U_1,U_2$ *standard* if $|U_j|$ for $j=1,2$ and $|U_1\cap U_2|$ are in the same ranges. Thus, all non-standard pairs make a negligible contribution in (3), and we only have to prove (4) for each standard pair $U_1,U_2$.

For a graph $H$ and a subset $U\subset V(H)$, let $E_H(U)$ denote the set of edges of $H$ with at least one vertex in $U$. Given two sets of edges $\mathcal{E}_1$ and $\mathcal{E}_2$ incident to the vertices in $U_1\setminus\{2\}$ and $U_2\setminus\{1\}$ respectively, let $G''=G''(U_1,U_2,\mathcal{E}_1,\mathcal{E}_2)$ be the random graph $G'$ subject to the conditions $E_{G'}(U_1\setminus\{2\})=\mathcal{E}_1$ and $E_{G'}(U_2\setminus\{1\})=\mathcal{E}_2$. Let $w''_k(i)=w_k^{G''}(i)$ denote the number of walks of length $k$ emanating from $i$ in $G''$ (the dependence of $w''_k(i)$ on $U_1,U_2,\mathcal{E}_1,\mathcal{E}_2$ is dropped for notational simplicity). Using this notation, we can write

$$\begin{aligned}p(U_1,U_2)=\sum_{\mathcal{E}_1,\mathcal{E}_2:\,\sum_{U_1}w'_1(i)=\sum_{U_2}w'_1(i)}&\mathsf{P}\left[E_{G'}(U_1\setminus\{2\})=\mathcal{E}_1,\,E_{G'}(U_2\setminus\{1\})=\mathcal{E}_2\right]\\&\times\mathsf{P}\left[\sum_{i\in U_1}w''_2(i)=\sum_{i\in U_2}w''_2(i)\right].\quad(5)\end{aligned}$$

We first show that

$$\sum_{\mathcal{E}_1,\mathcal{E}_2:\,\sum_{U_1}w'_1(i)=\sum_{U_2}w'_1(i)}\mathsf{P}\left[E_{G'}(U_1\setminus\{2\})=\mathcal{E}_1,\,E_{G'}(U_2\setminus\{1\})=\mathcal{E}_2\right]=O(1/n).\qquad(6)$$

Note that the sum in the left hand side of (6) is equal to the probability that $\sum_{i\in U_1}w'_1(i)=\sum_{i\in U_2}w'_1(i)$. This equality is equivalent to $\sum_{i\in U_1\setminus(U_2\cup\{2\})}w'_1(i)=\sum_{i\in U_2\setminus(U_1\cup\{1\})}w'_1(i)$, which in its turn is true if and only if $U_1\setminus(U_2\cup\{2\})$ and $U_2\setminus(U_1\cup\{1\})$ send the same number of edges to $[n]\setminus[(U_1\cup U_2\cup\{1,2\})\setminus(U_1\cap U_2)]$. Since the pair $U_1,U_2$ is standard, these numbers are independent binomial random variables with $\Theta(n^2)$ trials. Equality (6) now follows by Lemma 5.

We now can derive Equality (4) from Equality (6) by proving that

$$\mathsf{P}\left[\sum_{i\in U_1} w_2''(i) = \sum_{i\in U_2} w_2''(i)\right] = O(n^{-3/4}\ln^{1/4} n) \tag{7}$$

for each potential pair $\mathcal{E}_1, \mathcal{E}_2$. Again, it is enough to do this only for most probable pairs whose contribution in (5) is overwhelming. Specifically, let $w_2'(u,v)$ denote the number of all paths of length 2 between $u$ and $v$ in $G'$ and define

$$\Delta(i,j) = (w_2'(i,1) + w_2'(j,1)) - (w_2'(i,2) + w_2'(j,2)).$$

Note that the numbers $w_2'(i,1), w_2'(j,1), w_2'(i,2), w_2'(j,2)$ and, hence, the numbers $\Delta(i,j)$ are completely determined by specifying $E_{G'}(U_1 \setminus \{2\}) = \mathcal{E}_1$ and $E_{G'}(U_2 \setminus \{1\}) = \mathcal{E}_2$. We call a pair $\mathcal{E}_1, \mathcal{E}_2$ *standard* if $\Delta(i,j)$ takes on $O(\sqrt{n\ln n})$ different values for $i \neq j$ from $[n] \setminus (U_1 \cup U_2 \cup \{1,2\})$. The following fact shows that it is enough if we prove (7) for each standard pair $\mathcal{E}_1, \mathcal{E}_2$.

▷ Claim 6. If a pair $U_1, U_2$ is standard, then

$$|\{\Delta(i,j) \,:\, i,j \in [n] \setminus (U_1 \cup U_2 \cup \{1,2\}),\, i \neq j\}| = O(\sqrt{n\ln n})$$

with probability $1 - O(n^{-6})$.

Proof. Let $u_1 = |U_1|$, $u_2 = |U_2|$, and $u = |U_1 \cap U_2|$. Note that

$$\Delta(i,j) = |N_{G'}(i) \cap (U_1 \setminus U_2)| + |N_{G'}(j) \cap (U_1 \setminus U_2)| - |N_{G'}(i) \cap (U_2 \setminus U_1)| - |N_{G'}(j) \cap (U_2 \setminus U_1)|.$$

The four terms in the right hand side are independent random variables $\mathrm{Bin}(u_1 - u, 1/2)$, $\mathrm{Bin}(u_1 - u, 1/2)$, $\mathrm{Bin}(u_2 - u, 1/2)$, $\mathrm{Bin}(u_2 - u, 1/2)$ respectively. Since $N - \mathrm{Bin}(N,p) \sim \mathrm{Bin}(N, 1-p)$, we conclude that $\Delta(i,j) \sim 2u - 2u_2 + \mathrm{Bin}(2u_1 + 2u_2 - 4u, 1/2)$. The Chernoff bound (see Lemma 4) implies that, for each pair $i,j$, the inequalities

$$2u - 2u_2 + (u_1 + u_2 - 2u)\left(1 - \frac{\sqrt{2\ln n}}{\sqrt{u_1 + u_2 - 2u}}\right) \leq \Delta(i,j) \leq 2u - 2u_2 + (u_1 + u_2 - 2u)\left(1 + \frac{\sqrt{2\ln n}}{\sqrt{u_1 + u_2 - 2u}}\right)$$

are violated with probability at most $O(n^{-8})$. By the union bound, the probability that not all values $\Delta(i,j)$ fall in an integer interval of length at most

$$2\sqrt{2\ln n(u_1 + u_2 - 2u)} = O(\sqrt{n\ln n})$$

is bounded by $O(n^{-6})$. ◁

It remains to prove (7) for a fixed standard pair $\mathcal{E}_1, \mathcal{E}_2$. Note that all walks of length 3 starting from 1 and 2 and having at least 2 vertices inside $U_1 \cup U_2 \cup \{1,2\}$ are determined by $U_1, U_2, \mathcal{E}_1, \mathcal{E}_2$. Let $\gamma_j = \gamma_j(U_1, U_2, \mathcal{E}_1, \mathcal{E}_2)$ denote the number of such walks starting at $j$ for $j = 1, 2$. Let $e''_{i,j}$ be the indicator random variable of the presence of the edge $\{i,j\}$ in $G''$. The equality $\sum_{i\in U_1} w_2''(i) = \sum_{i\in U_2} w_2''(i)$ can be rewritten as

$$\gamma_1 + \sum_{i,j\notin U_1\cup U_2\cup\{1,2\}} e''_{ij}(w_2'(i,1) + w_2'(j,1)) = \gamma_2 + \sum_{i,j\notin U_1\cup U_2\cup\{1,2\}} e''_{ij}(w_2'(i,2) + w_2'(j,2)), \tag{8}$$

where the sums count the walks of length 3 from 1 and 2 whose last two vertices are outside $U_1 \cup U_2 \cup \{1,2\}$. Since $\mathcal{E}_1, \mathcal{E}_2$ is a standard pair, there exists an integer $x \neq 0$ such that $\Delta(i,j) = x$ for $\Omega(n^{3/2}/\sqrt{\ln n})$ pairs $i,j$. Let $S_x$ be the set of all such pairs. Let $G^* = G^*(U_1, U_2, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}^*)$ be obtained from $G''$ by exposing all edges except those between $i,j$ in $S_x$, where $\mathcal{E}^*$ is the set of exposed edges. Equality (8) is fulfilled if and only if

$$\sum_{\{i,j\} \in S_x} e''_{ij} x = \gamma(U_1, U_2, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}^*) \tag{9}$$

for some integer $\gamma(U_1, U_2, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}^*)$ which is completely determined by $U_1, U_2, \mathcal{E}_1, \mathcal{E}_2, \mathcal{E}^*$. It remains to note that the binomial random variable $\sum_{\{i,j\} \in S_x} e''_{ij} \sim \mathrm{Bin}(|S_x|, 1/2)$ takes on any fixed value with probability at most $\binom{|S_x|}{\lfloor |S_x|/2 \rfloor}/2^{|S_x|} = O(|S_x|^{-1/2}) = O(n^{-3/4} \ln^{1/4} n)$, where the first equality is due to (1). This completes the proof of Equality (7) and of the whole theorem.

▶ Remark 7. The probability bound in Theorem 1 cannot be significantly improved because $\mathsf{P}[\mathbf{w}_3^G(1) = \mathbf{w}_3^G(2)] = n^{-\Omega(1)}$. To see this, note first that $\mathsf{P}[w_1^G(1) = w_1^G(2)] = \Theta(n^{-1/2})$ (see the proof of Lemma 5). Assuming that a standard pair $U_1, U_2$ with $|U_1| = |U_2|$ is fixed, we can similarly show that $p(U_1, U_2) = \Theta(n^{-1})$, which implies that $\mathsf{P}[\big(w_1^G(1), w_2^G(1)\big) = \big(w_1^G(2), w_2^G(2)\big)] = \Theta(n^{-3/2})$. Showing a polynomial lower bound for $\mathsf{P}[\big(w_1^G(1), w_2^G(1), w_3^G(1)\big) = \big(w_1^G(2), w_2^G(2), w_3^G(2)\big)]$ is a slightly more delicate issue. Following the same proof strategy as for the upper bound, we have to ensure that the equation (9) has at least one integer solution $\sum_{\{i,j\} \in S_x} e''_{ij}$. We can do this because we have enough freedom in adjusting the right hand side of (9) by choosing an appropriate value of $\gamma_1 - \gamma_2$. Indeed, first of all, $|x|$ does not exceed $2n$ with probability 1. Second, we have an interval of length at least $100n$ for the values of $\gamma_1 - \gamma_2$ that are reachable with probability $\Omega(n^{-1})$. As easily seen, this is enough for obtaining a desired lower bound.

We leave as an open question whether Theorem 1 can be improved by excluding paths of length 3. Our conjecture is that this is impossible, that is, Theorem 1 is optimal in this respect, but proving this poses some technical challenges.

# 3 Comparing WM and CR

## 3.1 Color refinement

We begin with a formal description of the *color refinement* algorithm (*CR* for short). CR operates on vertex-colored graphs but applies also to uncolored graph by assuming that their vertices are colored uniformly. An input to the algorithm consists either of a single graph or a pair of graphs. Consider the former case first. For an input graph $G$ with initial coloring $C_0$, CR iteratively computes new colorings

$$C_i(x) = \Big(C_{i-1}(x), \{\!\{C_{i-1}(y)\}\!\}_{y \in N(x)}\Big), \tag{10}$$

where $\{\!\{\}\!\}$ denotes a multiset and $N(x)$ is the neighborhood of a vertex $x$. Denote the partition of $V(G)$ into the color classes of $C_i$ by $\mathcal{P}_i$. Note that each subsequent partition $\mathcal{P}_{i+1}$ is either finer than or equal to $\mathcal{P}_i$. If $\mathcal{P}_{i+1} = \mathcal{P}_i$, then $\mathcal{P}_j = \mathcal{P}_i$ for all $j \geq i$. Suppose that the color partition stabilizes in the $t$-th round, that is, $t$ is the minimum number such that $\mathcal{P}_t = \mathcal{P}_{t-1}$. CR terminates at this point and outputs the coloring $C = C_t$. Note that if the colors are computed exactly as defined by (10), they will require exponentially long color names. To prevent this, the algorithm renames the colors after each refinement step, using the same set of no more than $n$ color names.

We say that a graph $G$ is *CR-discrete* if $C(x) \neq C(x')$ for all $x \neq x'$.

If an input consists of two graphs $G$ and $H$, then it is convenient to assume that their vertex sets $V(G)$ and $V(H)$ are disjoint. The vertex colorings of $G$ and $H$ define an initial coloring $C_0$ of the union $V(G) \cup V(H)$, which is iteratively refined according to (10). The color partition $\mathcal{P}_i$ is defined exactly as above but now on the whole set $V(G) \cup V(H)$. As soon as the color partition of $V(G) \cup V(H)$ stabilizes[1], CR terminates and outputs the current coloring $C = C_t$ of $V(G) \cup V(H)$. The color names are renamed for both graphs synchronously.

We say that CR *distinguishes* $G$ and $H$ if $\{\!\{C(x)\}\!\}_{x \in V(G)} \neq \{\!\{C(x)\}\!\}_{x \in V(H)}$. A graph $G$ is called *CR-identifiable* if it is distinguishable by CR from every non-isomorphic $H$. Note that every CR-discrete graph is CR-identifiable.

## 3.2 Proof of Theorem 3

### 3.2.1 Parts 1 and 2

Parts 1 and 2 of Theorem 3 follow immediately from the lemma below. We prove this lemma by a direct combinatorial argument. Alternatively, one can use an algebraic approach in [11, Theorem 2] or the connection to finite variable logics exploited in [4, Lemma 4].

▶ **Lemma 8.** *Let $G$ and $H$ be uncolored $n$-vertex graphs (the case $G = H$ is not excluded). Let $x \in V(G)$, $x' \in V(H)$, and $k$ be an arbitrary non-negative integer. Then $C_k(x) \neq C_k(x')$ whenever $w_k^G(x) \neq w_k^H(x')$.*

**Proof.** Using the induction on $k$, we prove that $w_k^G(x) = w_k^H(x')$ whenever $C_k(x) = C_k(x')$. In the base case of $k = 0$, these equalities are equivalent just because they are both true by definition (recall that $w_0^G(x) = 0$). Assume that $C_k(y) = C_k(y')$ implies $w_k^G(y) = w_k^H(y')$ for all $y \in V(G)$ and $y' \in V(H)$. Let $C_{k+1}(x) = C_{k+1}(x')$. By the definition of the refinement step, we have $\{\!\{C_k(y)\}\!\}_{y \in N(x)} = \{\!\{C_k(y)\}\!\}_{y \in N(x')}$. Using the induction assumption, from here we derive the equality $\{\!\{w_k^G(y)\}\!\}_{y \in N(x)} = \{\!\{w_k^H(y)\}\!\}_{y \in N(x')}$. The equality $w_{k+1}^G(x) = w_{k+1}^H(x')$ now follows by noting that $w_{k+1}^G(x) = \sum_{y \in N(x)} w_k^G(y)$. ◀

### 3.2.2 Part 3

We now construct a graph $G$ with the three desired properties (a)–(c). Note that this graph can be used in an obvious way to produce infinitely many examples separating the strength of WM and CR.

Let $\mathbb{Z}_n$ denote the cyclic group with elements $0, 1, \ldots, n$ and operation being the addition modulo $n$. Our construction is based on the well-known Shrikhande graph; see, e.g., [12]. This is the Cayley graph of the group $\mathbb{Z}_4 \times \mathbb{Z}_4$ with connection set $\{\pm(1,0), \pm(0,1), \pm(1,1)\}$. A natural drawing of the Shrikhande graph on the torus can be seen in both parts of Fig. 1.

Recall that a graph $G$ is *strongly regular* with parameters $(n, d, \lambda, \mu)$ if it has $n$ vertices, every vertex in $G$ has $d$ neighbors (i.e., $G$ is *regular* of degree $d$), every two adjacent vertices of $G$ have $\lambda$ common neighbors, and every two non-adjacent vertices have $\mu$ common neighbors. We will use two properties of the Shrikhande graph:

- It is a strongly regular graph with parameters $(16, 6, 2, 2)$.
- The pairs $u, v$ of non-adjacent vertices in the graph are split into two categories depending on whether the two common neighbors of $u$ and $v$ are adjacent or not.

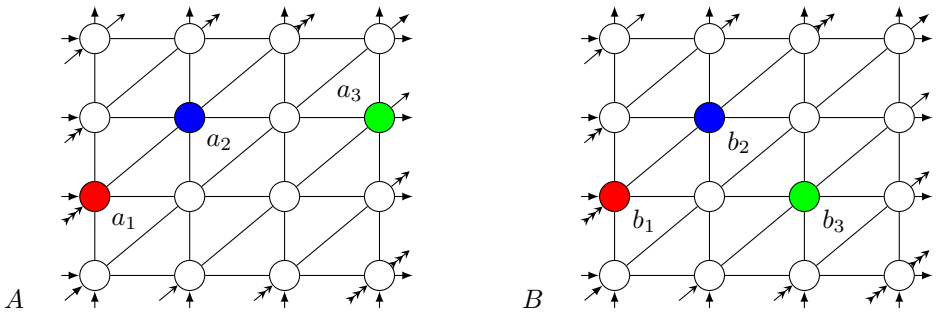



**Figure 1** Two colored versions of the Shrikhande graph.

Consider two copies $A$ and $B$ of the Shrikhande graph. In each of $A$ and $B$, let us individualize three vertices, $a_1, a_2, a_3$ in $A$ and $b_1, b_2, b_3$ in $B$, by assigning unique colors as shown in Fig. 1. The vertices $a_i$ and $b_i$ for each $i = 1, 2, 3$ are equally colored. All non-individualized vertices are considered also colored, all in the same color. Of the three vertices $a_1, a_2, a_3$, only $a_1$ and $a_2$ are adjacent, and the vertices $b_1, b_2, b_3$ have the same adjacency pattern. An important difference between $A$ and $B$ is that the two common neighbors of $b_2$ and $b_3$ are adjacent while the two common neighbors of $a_2$ and $a_3$ are not.[2] This implies that the vertex-colored graphs $A$ and $B$ are non-isomorphic.

Before presenting further details, we give a brief outline of the rest of the proof. We will begin with establishing some useful properties of $A$ and $B$. Though these graphs are non-isomorphic, it is useful to notice that they are still quite similar in the sense that they are indistinguishable by one round of CR (Claim 9). On the other hand, both $A$ and $B$ are CR-discrete (Claim 10) and are, therefore, distinguished after CR makes sufficiently many rounds (Claim 11). The desired graph $G$ will be constructed from $A$ and $B$ by connecting the equally colored vertices, i.e., $a_i$ and $b_i$, via new edges and vertices. While $a_1, a_2, a_3, b_1, b_2, b_3$ are not colored any more in $G$, their neighborhoods are modified so that their colors are actually simulated by iterated degrees. This allows us to derive from Claims 10 and 11 that $G$ is CR-discrete (Claim 12). On the other hand, $G$ is not WM-discrete (Claim 14). In order to show that some vertices in $G$ have the same numbers of outgoing walks of each length, we use some basic properties of strongly regular graphs (Claim 13) and the fact that a walk can leave $A$ or $B$ only via one of the vertices $a_1, a_2, a_3, b_1, b_2, b_3$ (and here an important role is played by Claim 9). Finally, we argue that $G$ is not WM-identifiable (Claim 15). Indeed, if we construct another graph $G'$ similarly to $G$ but using two copies of $A$, then $G$ and $G'$ will have the same walk matrix.

We now proceed with the detailed proof.

▷ Claim 9. After the first round of CR, the vertex-colored graphs $A$ and $B$ are still indistinguishable. That is, there is a bijection $f : V(A) \to V(B)$ such that $C_1(x) = C_1(f(x))$ for all $x \in V(A)$.

[1] Note that the stabilization on each of the sets $V(G)$ and $V(H)$ can occur earlier than on $V(G) \cup V(H)$.
[2] Using the fact that the Shrikhande graph is arc-transitive, it is easy to check that $A$ and $B$ are defined uniquely up to isomorphism of colored graphs.

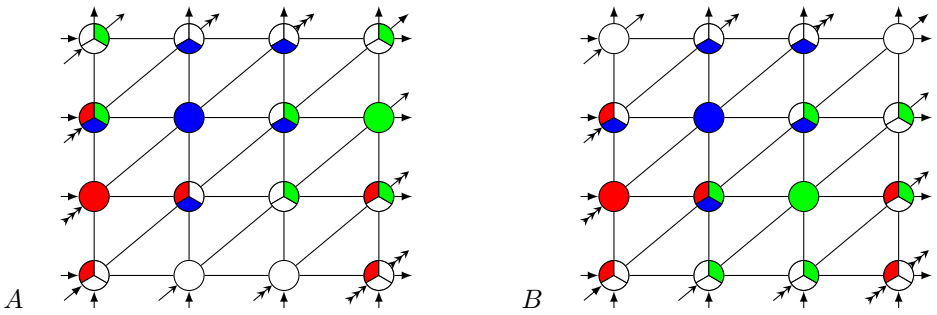



**Figure 2** The colorings of $A$ and $B$ after the first color refinement round. For each $i = 1, 2, 3$, the vertices $a_i$ and $b_i$ have the same unique color. The color of each non-individualized vertex is determined by its adjacency to the individualized vertices. For example, the color of a vertex in $A$ means that this vertex is adjacent to $a_1$ and $a_2$ but not to $a_3$.

Proof. Since $A$ and $B$ are regular graphs of the same degree, the equally colored vertices $x \in V(A)$ and $x' \in V(B)$ obtain distinct colors after the first color refinement round only when their neighborhoods contain different sets of individualized vertices (that is, $a_i \in N(x)$ while $b_i \notin N(x')$ or vice versa for some $i = 1, 2, 3$). This is not the case for the individualized vertices because the correspondence $a_i \mapsto b_i$ is a partial isomorphism of $A$ and $B$. As for the non-individualized vertices, both $A$ and $B$ have exactly one vertex adjacent to all the three individualized vertices, three vertices adjacent to exactly two of them, and two vertices adjacent to none of them. Moreover, in both $A$ and $B$ there are two non-individualized vertices adjacent to $a_i$ (resp. to $b_i$) for each $i = 1, 2$ and three non-individualized vertices adjacent to $a_3$ (resp. to $b_3$). The colorings of $A$ and $B$ after the first refinement round are shown in Fig. 2. ◁

▷ Claim 10. Both vertex-colored graphs $A$ and $B$ are CR-discrete.

Proof. Call a vertex *solitary* if CR colors it differently than the other vertices of the graphs. We prove that every vertex in $A$ is solitary. Virtually the same argument applies also to $B$. The individualized vertices $a_1, a_2, a_3$ are solitary from the very beginning. The single vertex $a$ adjacent to all of them is obviously also solitary. Thus, $A$ contains a triangle subgraph whose all vertices, namely $a, a_1, a_2$, are solitary. Let $a'$ be the common neighbor of $a_1$ and $a_2$ different from $a$ (recall that the Shrikhande graph is strongly regular with the third parameter $\lambda = 2$). The fact that $a_1$ and $a_2$ are solitary implies that the equality $C(a') = C(x)$ for $x \neq a'$ can be true only if $x = a$, which is actually impossible because $a$ is solitary. Therefore, $a'$ is solitary too. This argument applies to any triangle whose all vertices are solitary and to the other common neighbor of any two vertices of this triangle. Consider the graph whose vertices are the triangles of the Shrikhande graph, adjacent exactly when they share an edge. This graph (known as the *Dyck graph*) is obviously connected, which readily implies that all vertices of $A$ are solitary. ◁

▷ Claim 11. The vertex-colored graphs $A$ and $B$ are distinguishable by CR.

Proof. Recall that $A$ and $B$ are non-isomorphic because the two common neighbors of $b_2$ and $b_3$ are adjacent while the two common neighbors of $a_2$ and $a_3$ are not. By Claim 10, both $A$ and $B$ are CR-discrete. Assume that $A$ and $B$ are indistinguishable by CR. Let $f$ be

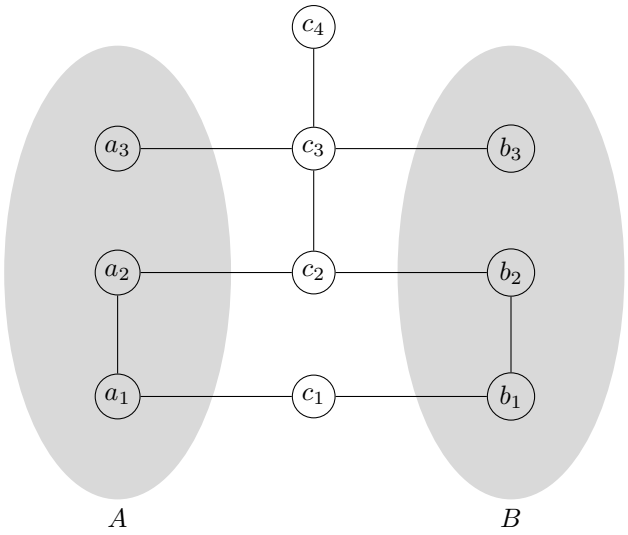



**Figure 3** Construction of $G$.

the bijection from $V(A)$ to $V(B)$ respecting the final CR-coloring, that is, $C(f(x)) = C(x)$ for all $x \in V(A)$. Since $f$ is not an isomorphism, there are vertices $u$ and $v$ in $A$ such that $u$ and $v$ are adjacent but $f(u)$ and $f(v)$ are not or vice versa. This shows, however, that the coloring $C$ is still unstable because, by the refinement rule, $u$ and $f(u)$ have to receive distinct colors in the next round. This contradiction proves the claim. ◁

We now construct a graph $G$ as the vertex-disjoint union of $A$ and $B$ where each pair $a_i, b_i$ is connected via new edges and new intermediate vertices as shown in Fig. 3. Thus, $V(G) = V(A) \cup V(B) \cup \{c_1, c_2, c_3, c_4\}$ where $c_1, c_2, c_3, c_4$ are new connector vertices of degree $2, 3, 4, 1$ respectively. The graph $G$ is uncolored, that is, the colors of the six individualized vertices $a_1, a_2, a_3, b_1, b_2, b_3$ are erased. The next claim proves Part 3(a) of the theorem.

▷ Claim 12. $G$ is CR-discrete.

Proof. The connector vertices $c_1, c_2, c_3, c_4$ have unique degrees $2, 3, 4, 1$ and become solitary after the first refinement round. The vertices $a_1, a_2, a_3$ have degree 7, while the other vertices in $A$ have degree 6. Each of the three vertices $a_1, a_2, a_3$ is distinguished from the other two by the adjacency to its own connector. It follows that after the second refinement round, the colors $C_2(a_1), C_2(a_2), C_2(a_3)$ become unique within $A$ (even when still $C_2(a_i) = C_2(b_i)$). Claim 10, therefore, implies that eventually $C(x) \neq C(x')$ for all $x \neq x'$ in $A$. The same argument applies to $B$. Using the same argument as in the proof of Claim 11, we also have $C(x) \neq C(x')$ for all $x \in V(A)$ and $x' \in V(B)$. ◁

Let $w_k^R(x, y)$ denote the number of walks of length $k$ from a vertex $x$ to a vertex $y$ in a graph $R$. We will need the following simple and well-known facts.[3]

[3] Let $P_{k+1}$ be a path of length $k$ with end vertices $s$ and $t$. Note that $w_k^R(x, y)$ is equal to the number of all homomorphisms from $P_{k+1}$ to $R$ taking $s$ to $x$ and $t$ to $y$. Part 2 of Claim 13 is a particular case of a much more general result about the invariance of homomorphism counts under the Weisfeiler-Leman equivalence for graphs with designated vertices [4, Lemma 4].

▷ Claim 13.

1. If $R$ is a regular graph of degree $d$, then $w_k^R(x) = d^k$ for every $x \in V(R)$.
2. Suppose now that $R$ is a strongly regular graph with parameters $(n, d, \lambda, \mu)$ and fix an arbitrary $k \geq 0$. Then the walk count $w_k^R(x, x)$ is the same for every $x \in V(R)$. If $x \neq y$, then the value of $w_k^R(x, y)$ depends only on the adjacency of $x$ and $y$ (and on the parameters $d, \lambda, \mu$).

Proof. Part 1 is obvious. Part 2 follows from an easy inductive argument. Indeed, it is trivially true for $k = 0$. Assume that $w_k^R(x, y) = a_k$ for all adjacent $x$ and $y$ and that $w_k^R(x, y) = n_k$ for all non-adjacent unequal $x$ and $y$. Then $w_{k+1}^R(x, x) = \sum_{z \in N(x)} w_k^R(z, x) = d\, a_k$. If $x$ and $y$ are adjacent, then

$$w_{k+1}^R(x, y) = \sum_{z \in N(x) \cap N(y)} w_k^R(z, y) + \sum_{z \in N(x) \setminus N(y)} w_k^R(z, y) = \lambda a_k + (d - \lambda) n_k.$$

If $x$ and $y$ are non-adjacent and unequal, then

$$w_{k+1}^R(x, y) = \sum_{z \in N(x) \cap N(y)} w_k^R(z, y) + \sum_{z \in N(x) \setminus N(y)} w_k^R(z, y) = \mu a_k + (d - \mu) n_k,$$

enabling the induction step. ◁

We are now prepared to prove Part 3(b) of the theorem.

▷ Claim 14. $G$ is not WM-discrete.

Proof. Define an equivalence relation $\equiv$ on $V(G)$ as follows. Each connector vertex is equivalent only to itself. Let $C_1$ be the coloring of $V(A) \cup V(B)$ obtained after the first round of the execution of CR on the vertex-colored graphs $A$ and $B$; see Claim 9. We set $x \equiv x'$ for $x, x' \in V(A) \cup V(B)$ if $C_1(x) = C_1(x')$. Recall that the largest equivalence class of $\equiv$ consists of six vertices (three uncolored vertices in $A$ adjacent to $a_3$ but neither to $a_1$ nor to $a_2$ and three uncolored vertices in $B$ adjacent to $b_3$ but neither to $b_1$ nor to $b_2$). We claim that $w_k^G(x) = w_k^G(x')$ for every $k$ whenever $x \equiv x'$. Indeed, if $x \in V(A)$, then

$$w_k^G(x) = w_k^A(x) + \sum_{i=1}^{3} \sum_{j=0}^{k-1} w_j^A(x, a_i) w_{k-j-1}^G(c_i). \tag{11}$$

Here, we separately consider the walks of length $k$ inside $A$ and the walks of length $k$ leaving $A$. A walk can leave $A$ only after visiting one of the vertices $a_1, a_2, a_3$. If such a walk leaves $A$ first after the $j$-th step via $a_i$, it arrives at the connector $c_i$ and, starting from it, makes the remaining $k - j - 1$ steps. The similar equality holds for $x \in V(B)$.

It remains to notice that the right hand side of (11) and its analog for $B$ yield the same value for all $x$ in the same $\equiv$-class. Indeed, let $x \equiv x'$ and suppose that $x \in A$ and $x' \in B$ (the cases $x, x' \in A$ and $x, x' \in B$ are completely similar). Then $w_k^A(x) = w_k^B(x') = 6^k$ by Part 1 of Claim 13. Finally, for each $j$ the equalities $w_j^A(x, a_i) = w_j^B(x, b_i)$ for $i = 1, 2, 3$ follow from Part 2 of Claim 13 by the definition of the relation $\equiv$ and the description of $C_1$ in the proof of Claim 9. ◁

It remains to prove Part 3(c) of the theorem.

▷ Claim 15. $G$ is not WM-identifiable.

Proof. Construct $G'$ in the same way as $G$ but using a copy $A'$ of $A$ instead of $B$. The graphs $G$ and $G'$ are non-isomorphic, basically because $A$ and $B$ are non-isomorphic as colored graphs. In particular, $G'$ has an automorphism fixing the connector vertices and transposing $A$ and $A'$, whereas $G$ has no non-trivial automorphism by Claim 12. Fix a colored-graph isomorphism $f'$ from $A'$ to $A$. Define a bijection $F$ from $V(G') = V(A) \cup V(A') \cup \{c_1, c_2, c_3, c_4\}$ onto $V(G) = V(A) \cup V(B) \cup \{c_1, c_2, c_3, c_4\}$ so that $F(c_i) = c_i$ for $i = 1, 2, 3, 4$, the restriction $f$ of $F$ to $V(A)$ is as in Claim 9, and the restriction of $F$ to $V(A')$ is the isomorphism $f'$. The proof of Claim 14 applies to the graph $G'$ virtually without changes. In particular, the analog of Equality (11) for $G'$ allows us to show by a simple induction that $w_k^{G'}(x) = w_k^G(f(x))$ for $x \in V(A)$ and $w_k^{G'}(x') = w_k^G(f'(x'))$ for $x' \in V(A')$, as well as that $w_k^{G'}(c_i) = w_k^G(c_i)$ for $i = 1, 2, 3, 4$. Thus, for every $x \in V(G')$ we have $w_k^{G'}(x) = w_k^G(F(x))$ for all $k$, implying that $G$ and $G'$ are WM-indistinguishable. ◁

The proof of Theorem 3 is complete.

## References


1. Noga Alon and Joel H. Spencer. *The probabilistic method*. John Wiley & Sons, 2016.
2. László Babai, Paul Erdős, and Stanley M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980.
3. László Babai and Ludek Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (FOCS'79)*, pages 39–46. IEEE Computer Society, 1979. doi:10.1109/SFCS.1979.8.
4. Z. Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010.
5. Chris Godsil. Controllable subsets in graphs. *Ann. Comb.*, 16(4):733–744, 2012. doi:10.1007/s00026-012-0156-3.
6. Elias M. Hagos. Some results on graph spectra. *Linear Algebra Appl.*, 356(1-3):103–111, 2002. doi:10.1016/S0024-3795(02)00324-5.
7. David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021. doi:10.4007/annals.2021.193.2.4.
8. Fenjin Liu and Johannes Siemons. Unlocking the walk matrix of a graph. *J. Algebr. Comb.*, 55(3):663–690, 2022. doi:10.1007/s10801-021-01065-3.
9. H. L. Morgan. The generation of a unique machine description for chemical structures — a technique developed at chemical abstracts service. *J. Chem. Doc.*, 5(2):107–113, 1965. doi:10.1021/c160017a018.
10. Sean O'Rourke and Behrouz Touri. On a conjecture of Godsil concerning controllable random graphs. *SIAM J. Control. Optim.*, 54(6):3347–3378, 2016. doi:10.1137/15M1049622.
11. David L. Powers and Mohammad M. Sulaiman. The walk partition and colorations of a graph. *Linear Algebra Appl.*, 48:145–159, 1982. doi:10.1016/0024-3795(82)90104-5.
12. Sharad S. Sane. The Shrikhande graph. *Resonance*, 20:903–918, 2015. doi:10.1007/s12045-015-0255-7.

# Improved Algorithms for Distance Selection and Related Problems

**Haitao Wang**
School of Computing, University of Utah, Salt Lake City, UT, USA

**Yiming Zhao**[1]
Department of Computer Science, Utah State University, Logan, UT, USA

## Abstract

In this paper, we propose new techniques for solving geometric optimization problems involving interpoint distances of a point set in the plane. Given a set $P$ of $n$ points in the plane and an integer $1 \leq k \leq \binom{n}{2}$, the distance selection problem is to find the $k$-th smallest interpoint distance among all pairs of points of $P$. The previously best deterministic algorithm solves the problem in $O(n^{4/3} \log^2 n)$ time [Katz and Sharir, 1997]. In this paper, we improve their algorithm to $O(n^{4/3} \log n)$ time. Using similar techniques, we also give improved algorithms on both the two-sided and the one-sided discrete Fréchet distance with shortcuts problem for two point sets in the plane. For the two-sided problem (resp., one-sided problem), we improve the previous work [Avraham, Filtser, Kaplan, Katz, and Sharir, 2015] by a factor of roughly $\log^2(m+n)$ (resp., $(m+n)^\epsilon$), where $m$ and $n$ are the sizes of the two input point sets, respectively. Other problems whose solutions can be improved by our techniques include the reverse shortest path problems for unit-disk graphs. Our techniques are quite general and we believe they will find many other applications in future.

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

**Keywords and phrases** Geometric optimization, distance selection, Fréchet distance, range searching

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.101

**Related Version** *Full Version*: https://arxiv.org/abs/2306.01073

**Funding** This research was supported in part by NSF under Grants CCF-2005323 and CCF-2300356.

## 1 Introduction

In this paper, we propose new techniques for solving geometric optimization problems involving interpoint distances in a point set in the plane. More specifically, the optimal objective value of these problems is equal to the (Euclidean) distance of two points in the set. Our techniques usually yield improvements over the previous work by at least a logarithmic factor (and sometimes a polynomial factor).

The first problem we consider is the *distance selection* problem: Given a set $P$ of $n$ points in the plane and an integer $1 \leq k \leq \binom{n}{2}$, the problem asks for the $k$-th smallest interpoint distance among all pairs of points of $P$. The problem can be easily solved in $O(n^2)$ time. The first subquadratic time algorithm was given by Chazelle [10]; the algorithm runs in $O(n^{9/5} \log^{4/5} n)$ time and is based on Yao's technique [21]. Later, Agarwal, Aronov, Sharir, and Suri [1] gave a better algorithm of $O(n^{3/2} \log^{5/2} n)$ time and subsequently Goodrich [13] solved the problem in $O(n^{4/3} \log^{8/3} n)$ time. Katz and Sharir [14] finally presented an $O(n^{4/3} \log^2 n)$ time algorithm. All above are deterministic algorithms. Several randomized algorithms have also been proposed for the problem. The randomized algorithm of [1] runs in $O(n^{4/3} \log^{8/3} n)$ expected time. Matousek [17] gave another randomized algorithm of

[1] Corresponding author.


$O(n^{4/3}\log^{2/3} n)$ expected time. Very recently, Chan and Zheng proposed a randomized algorithm of $O(n^{4/3})$ expected time (see the arXiv version of [9]). Also, the time complexity can be made as a function of $k$. In particular, Chan's randomized techniques [7] solved the problem in $O(n\log n + n^{2/3}k^{1/3}\log^{5/3} n)$ expected time and Wang [18] recently improved the algorithm to $O(n\log n + n^{2/3}k^{1/3}\log n)$ expected time; these algorithms are particularly interesting when $k$ is relatively small.

In this paper, we present a new deterministic algorithm that solves the distance selection problem in $O(n^{4/3}\log n)$ time. Albeit slower than the randomized algorithm of Chan and Zheng [9], our algorithm is the first progress on the deterministic solution since the work of Katz and Sharir [14] published 25 years ago (30 years if we consider their conference version in SoCG 1993). One technique we introduce is an algorithm for solving the following *partial batched range searching problem*.

▶ **Problem 1** (Partial batched range searching). *Given a set $A$ of $m$ points and a set $B$ of $n$ points in the plane and an interval $(\alpha,\beta]$, one needs to construct two collections of edge-disjoint complete bipartite graphs $\Gamma(A,B,\alpha,\beta) = \{A_t \times B_t \mid A_t \subseteq A, B_t \subseteq B\}$ and $\Pi(A,B,\alpha,\beta) = \{A'_s \times B'_s \mid A'_s \subseteq A, B'_s \subseteq B\}$ such that the following two conditions are satisfied.*

1. *For each pair $(a,b) \in A_t \times B_t \in \Gamma(A,B,\alpha,\beta)$, the (Euclidean) distance $\|ab\|$ between points $a \in A_t$ and $b \in B_t$ is in $(\alpha,\beta]$.*
2. *For any two points $a \in A$ and $b \in B$ with $\|ab\| \in (\alpha,\beta]$, either $\Gamma(A,B,\alpha,\beta)$ has a unique graph $A_t \times B_t$ that contains $(a,b)$ or $\Pi(A,B,\alpha,\beta)$ has a unique graph $A'_s \times B'_s$ that contains $(a,b)$.*

*In other words, the two collections $\Gamma$ and $\Pi$ together record all pairs $(a,b)$ of points $a \in A$ and $b \in B$ whose distances are in $(\alpha,\beta]$. While all pairs of points recorded in $\Gamma$ have their distances in $(\alpha,\beta]$, this may not be true for $\Pi$. For this reason, we sometimes call the point pairs recorded in $\Pi$* uncertain pairs.

Note that if context is clear, we sometimes use $\Gamma$ and $\Pi$ to refer to $\Gamma(A,B,\alpha,\beta)$ and $\Pi(A,B,\alpha,\beta)$, respectively. Also, for short, we use BRS to refer to batched range searching.

In the traditional BRS, which has been studied with many applications, e.g.,[20, 15, 4], the collection $\Pi$ is $\emptyset$ (and thus $\Gamma$ itself satisfies the two conditions in Problem 1); for differentiation, we refer to this case as the *complete BRS*. The advantage of the partial problem over the complete problem is that the partial problem can usually be solved faster, with a sacrifice that some uncertain pairs (i.e., those recorded in $\Pi$) are left unresolved. As will be seen later, in typical applications the number of those uncertain pairs can be made small enough so that they can be handled easily without affecting the overall runtime of the algorithm. More specifically, we derive an algorithm to compute a solution for the partial BRS, whose runtime is controlled by a parameter (roughly speaking, the runtime increases as the graph sizes of $\Pi$ decreases). Previously, Katz and Sharir [14] gave an algorithm for the complete problem. Our solution, albeit for the more general partial problem, even improves their algorithm by roughly a logarithmic factor when applied to the complete case.

On the one hand, our partial BRS solution helps achieve our new result for the distance selection problem. On the other hand, combining some techniques for the latter problem, we propose a general algorithmic framework that can be used to solve any geometric optimization problem that involves interpoint distances of a set of points in the plane. Consider such a problem whose optimal objective value (denoted by $\delta^*$) is equal to the distance of two points of a set $P$ of $n$ points in the plane. Assume that the decision problem (i.e., given $\delta$, decide whether $\delta \geq \delta^*$) can be solved in $T_D$ time. A straightforward algorithm for computing $\delta^*$ is to

use the distance selection algorithm and the decision algorithm to perform binary search on interpoint distances of all pairs of points of $P$; the algorithm runs in $O(\log n)$ iterations and each iteration takes $O(n^{4/3}\log n + T_D)$ time (if we use our new distance selection algorithm). As such, the total runtime is $O((n^{4/3}\log n + T_D)\log n)$. Using our new framework, the runtime can be bounded by $O((n^{4/3} + T_D)\log n)$, which is faster when $T_D = o(n^{4/3}\log n)$.

One application of this new framework is the *two-sided discrete Fréchet distance with shortcuts* problem, or *two-sided DFD* for short. Fréchet distance is used to measure the similarity between two curves and many of its variations have been studied, e.g., [2, 3, 4, 5, 6, 12]. To reduce the impact of outliers between two (sampled) curves, discrete Fréchet distance with shortcuts was proposed [4, 12]. If outliers of only one curve need to be taken care of, it is called *one-sided DFD*; otherwise it is *two-sided DFD*. Avraham, Filtser, Kaplan, Katz, and Sharir [4] solved the two-sided DFD in $O((m^{2/3}n^{2/3} + m + n)\log^3(m+n))$, where $m$ and $n$ are the numbers of vertices of the two input curves, respectively. Using our new framework, we improve their algorithm to $O((m^{2/3}n^{2/3}\cdot 2^{O(\log^*(m+n))} + m\log n + n\log m)\log(m+n))$ time, an improvement of roughly $O(\log^2(m+n))$.

For the one-sided DFD, the authors of [4] gave a randomized algorithm of $O((m+n)^{6/5+\epsilon})$ expected time, for any constant $\epsilon > 0$. Using our solution to the partial BRS, we improve their algorithm to $O((m+n)^{6/5}\log^{8/5}(m+n))$ expected time. Based on the techniques of [4], Katz and Sharir [15] proposed an algorithmic framework for solving geometric optimization problems that involve interpoint distances in a point set. Consider such a problem whose optimal objective value (denoted by $\delta^*$) is equal to the distance of two points of a set $P$ of $n$ points in the plane. The framework has two main procedures. The first procedure is to compute an interval that contains $\delta^*$ and with high probability at most $L$ interpoint distances of $P$. Using the interval and a *bifurcation tree* technique, the second main procedure finally computes $\delta^*$. Assuming that the decision problem can be solved in $T_D$ time, the first main procedure takes $O(n^{4/3+\epsilon}/L^{1/3} + T_D\cdot\log n\cdot\log\log n)$ expected time and the second one runs in $O(L^{1/2}\cdot T_D\cdot\log n)$ time, resulting in an algorithm of $O(n^{4/3+\epsilon}/L^{1/3} + T_D\cdot\log n\cdot\log\log n + L^{1/2}\cdot T_D\cdot\log n)$ expected time in total [4, 15]. Using our partial BRS solution, we improve the first main procedure to $O(n^{4/3}/L^{1/3}\cdot\log^2 n + T_D\cdot\log n\cdot\log\log n)$ expected time, which eliminates the $O(n^\epsilon)$ factor. Thus, the total expected time of the framework becomes $O(n^{4/3}/L^{1/3}\cdot\log^2 n + T_D\cdot\log n\cdot\log\log n + L^{1/2}\cdot T_D\cdot\log n)$. Our result for the one-sided DFD is a direct application of this framework. More specifically, since $T_D = O(m+n)$ [4], we set $L = (m+n)^{2/5}\log^{6/5}(m+n)$ and replace $n$ by $(m+n)$ in the above time complexity as there are two parameters $m$ and $n$ in the problem.

We demonstrate two more applications of the framework where our new techniques lead to improved results over the previous work: the *reverse shortest paths in unit-disk graphs* and its weighted case. Given a set $P$ of $n$ points in the plane and a parameter $\delta > 0$, the unit-disk graph $G_\delta(P)$ is an undirected graph whose vertex set is $P$ such that an edge connects two points $p, q \in P$ if the (Euclidean) distance between $p$ and $q$ is at most $\delta$. In the unweighted (resp., weighted) case, the weight of each edge is equal to 1 (resp., the distance between the two vertices). Given $P$, two points $s, t \in P$, and a parameter $\lambda$, the problem is to compute the smallest $\delta^*$ such that the shortest path length between $s$ and $t$ in $G_{\delta^*}(P)$ is at most $\lambda$.

Deterministic algorithms of $O(n^{5/4}\log^{7/4} n)$ and $O(n^{5/4}\log^{5/2} n)$ times are known for the unweighted and weighted problems, respectively [20]. The decision problem for the unweighted case can be solved in $O(n)$ time (after points of $P$ are sorted) [8] while the weighted case can be solved in $O(n\log^2 n)$ time [19]. As such, using their framework, Katz and Sharir [15] solved both problems in $O(n^{6/5+\epsilon})$ expected time (by setting $L = n^{2/5}$). With our improvement to the framework, we can now solve the unweighted problem in $O(n^{6/5}\log^{8/5} n)$ expected time (by setting $L = n^{2/5}\log^{6/5} n$) and solve the weighted case in $O(n^{6/5}\log^{12/5} n)$ expected time (by setting $L = n^{2/5}/\log^{6/5} n$).

In summary, we propose two algorithmic frameworks for geometric optimization problems that involve interpoint distances in a set of points in the plane. The first one is deterministic while the second one is randomized. The first framework is mainly useful when the decision algorithm time $T_D$ is relatively large (e.g., close to $O(n^{4/3})$) while the second one is more interesting when $T_D$ is small (e.g., near linear). Both frameworks rely on our solution to the partial BRS problem. As optimization problems involving interpoint distances are very common in computational geometry, we believe our techniques will find more applications.

**Outline.** The rest of the paper is organized as follows. Section 2 presents our algorithm for the partial BRS. The distance selection algorithm is described in Section 3. The two-sided DFD problem is solved in Section 4, where we also propose our first algorithmic framework. The one-sided DFD and our second algorithmic framework are discussed in Section 5. Due to the space limit, some details and proofs are omitted but can be found in the full paper.

## 2 Partial batched range searching

In this section, we present our solution to the partial BRS problem, i.e., Problem 1. We follow the notation in the statement of Problem 1. In particular, $m = |A|$ and $n = |B|$.

For any set $P$ of points and a compact region $R$ in the plane, let $P(R)$ denote the subset of points of $P$ in $R$, i.e., $P(R) = P \cap R$. For any point $p$ in the plane, with respect to the interval $(\alpha, \beta]$ in Problem 1, let $D_p$ denote the annulus centered at $p$ and having radii $\alpha$ and $\beta$ (e.g., see Fig. 1); so $D_p$ has an inner boundary circle of radius $\alpha$ and an outer boundary circle of radius $\beta$. We assume that $D_p$ includes its outer boundary circle but not its inner boundary circle. In this way, a point $q$ is in $D_p$ if and only if $\|pq\| \in (\alpha, \beta]$. Define $\mathcal{D}$ as the set of all annuli $D_p$ for all points $p \in A$. Define $\mathcal{C}$ to be the set of boundary circles of all annuli of $\mathcal{D}$. Hence, $\mathcal{C}$ consists of $2m$ circles. For any compact region $R$ in the plane, let $\mathcal{C}_R$ denote the subset of circles of $\mathcal{C}$ that intersect the relative interior of $R$.

An important tool we use is the cuttings [11]. For a parameter $1 \leq r \leq n$, a $(1/r)$-cutting $\Xi$ of size $O(r^2)$ for $\mathcal{C}$ is a collection of $O(r^2)$ constant-complexity cells whose union covers the plane such that the interior of each cell $\sigma \in \Xi$ is intersected by at most $m/r$ circles in $\mathcal{C}$, i.e., $|\mathcal{C}_\sigma| \leq m/r$. We actually use *hierarchical cuttings* [11]. We say that a cutting $\Xi'$ *c-refines* a cutting $\Xi$ if each cell of $\Xi'$ is contained in a single cell of $\Xi$ and every cell of $\Xi$ contains at most $c$ cells of $\Xi'$. Let $\Xi_0$ denote the cutting whose single cell is the whole plane. Then we define cuttings $\{\Xi_0, \Xi_1, ..., \Xi_k\}$, in which each $\Xi_i$, $1 \leq i \leq k$, is a $(1/\rho^i)$-cutting of size $O(\rho^{2i})$ that $c$-refines $\Xi_{i-1}$, for two constants $\rho$ and $c$. By setting $k = \lceil \log_\rho r \rceil$, the last cutting $\Xi_k$ is a $(1/r)$-cutting. The sequence $\{\Xi_0, \Xi_1, ..., \Xi_k\}$ of cuttings is called a hierarchical $(1/r)$-cutting of $\mathcal{C}$. For a cell $\sigma'$ of $\Xi_{i-1}$, $1 \leq i \leq k$, that fully contains cell $\sigma$ of $\Xi_i$, we say that $\sigma'$ is the *parent* of $\sigma$ and $\sigma$ is a *child* of $\sigma'$. Thus the hierarchical $(1/r)$-cutting can be viewed as a tree structure with $\Xi_0$ as the root.

A hierarchical $(1/r)$-cutting of $\mathcal{C}$ can be computed in $O(mr)$ time, e.g., by the algorithm in [18], which adapts Chazelle's algorithm [11] for hyperplanes. The algorithm also produces the subset $\mathcal{C}_\sigma$ for all cells $\sigma \in \Xi_i$ for all $i = 0, 1, \ldots, k$, implying that the total size of these subsets is $O(mr)$. In particular, each cell of the cutting produced by the algorithm of [18] is a *pseudo-trapezoid* that is bounded by two vertical line segments from left and right, an arc of a circle of $\mathcal{C}$ from top, and an arc of a circle of $\mathcal{C}$ from bottom (e.g., see Fig. 2).

Using cuttings, we obtain the following solution to the partial BRS problem.

▶ **Lemma 1.** *For any $r$ with $1 \leq r \leq \min\{m^{1/3}, n^{1/3}\}$, we can compute in $O(mr \log r + nr)$ time two collections $\Gamma(A, B, \alpha, \beta) = \{A_t \times B_t \mid A_t \subseteq A, B_t \subseteq B\}$ and $\Pi(A, B, \alpha, \beta) = \{A'_s \times B'_s \mid A'_s \subseteq A, B'_s \subseteq B\}$ of edge-disjoint complete bipartite graphs that satisfy the conditions*

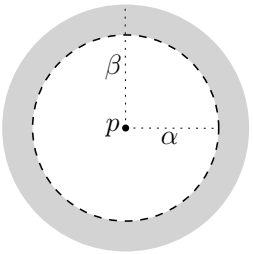



**Figure 1** An annulus $D_p$ (the grey region).

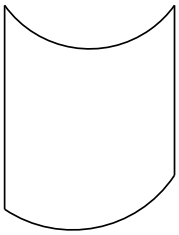


**Figure 2** Illustrating a pseudo-trapezoid.

*of Problem 1, with the following complexities: (1)* $|\Gamma| = O(r^4)$*; (2)* $\sum_t |A_t|, \sum_t |B_t| = O(mr \log r + nr)$*; (3)* $|\Pi| = O(r^4)$*; (4)* $|A'_s| = O(m/r^3)$ *and* $|B'_s| = O(n/r^3)$ *for each* $A'_s \times B'_s \in \Pi$*; (5) the number of pairs of points recorded in* $\Pi$ *is* $O(r^4 \cdot m/r^3 \cdot n/r^3) = O(mn/r^2)$.

**Proof.** We begin with constructing a hierarchical $(1/r)$-cutting $\{\Xi_0, \Xi_1, ..., \Xi_k\}$ for $\mathcal{C}$, which takes $O(mr)$ time as discussed above. We use $\Xi$ to refer to the set of all cells $\sigma$ in all cuttings $\Xi_i$, $0 \leq i \leq k$. Next we compute the set $B(\sigma)$ for each cell $\sigma$ in the cutting (recall that $B(\sigma)$ refers to the subset of points of $B$ inside $\sigma$; we call $B(\sigma)$ a *canonical subset*). This can be done in $O(n \log r)$ time in a top-down manner by processing each point of $B$ individually. Specifically, for each point $p \in B$, suppose we know that $p$ is in $\sigma'$ for a cell $\sigma'$ in $\Xi_{i-1}$ (which is true initially when $i = 1$ as $\Xi_0$ has a single cell that is the entire plane). By examining each child of $\sigma'$ we can find in $O(1)$ time the cell $\sigma$ of $\Xi_i$ that contains $p$ and then we add $p$ to $B(\sigma)$. Since $k = \Theta(\log r)$, each point of $B$ is stored in $O(\log r)$ canonical subsets and the total size of all canonical subsets $B(\sigma)$ for all cells $\sigma \in \Xi$ is $O(n \log r)$.

Next, for each cell $\sigma$ of $\Xi$, we compute another canonical subset $A_\sigma \subseteq A$. Specifically, a point $p \in A$ is in $A_\sigma$ if the annulus $D_p$ contains $\sigma$ but not $\sigma$'s parent. The subsets $A_\sigma$ for all cells $\sigma$ of $\Xi$ can be computed in $O(mr)$ time. Indeed, recall that the cutting algorithm already computes $\mathcal{C}_\sigma$ for all cells $\sigma \in \Xi$. For each $\Xi_{i-1}$, $1 \leq i \leq k$, for each cell $\sigma'$ of $\Xi_{i-1}$, we consider each circle $C \in \mathcal{C}_{\sigma'}$. Let $p$ be the point of $A$ such that $C$ is a bounding circle of the annulus $D_p$. For each child $\sigma$ of $\sigma'$, if $D_p$ fully contains $\sigma$, then we add $p$ to $A_\sigma$. In this way, $A_\sigma$ for all cells $\sigma$ of $\Xi$ can be computed in $O(mr)$ time since $\sum_{0 \leq i \leq k} \sum_{\sigma' \in \Xi_i} |\mathcal{C}_{\sigma'}| = O(mr)$ and each cell $\sigma'$ has $O(1)$ children. As such, the total size of $A_\sigma$ for all cells $\sigma \in \Xi$ is $O(mr)$.

By definition, for each cell $\sigma \in \Xi$, for any point $a \in A_\sigma$ and any point $b \in B(\sigma)$, we have $\|ab\| \in (\alpha, \beta]$. As such, we return $\{A_\sigma \times B(\sigma) \mid \sigma \in \Xi\}$ as a subcollection of $\Gamma(A, B, \alpha, \beta)$ to be computed for the lemma. Note that the complete bipartite graphs of $\{A_\sigma \times B(\sigma) \mid \sigma \in \Xi\}$ are edge-disjoint. The size of the subcollection is equal to the number of cells of the hierarchical cutting, which is $O(r^2)$. Also, we have shown above that $\sum_{\sigma \in \Xi} |A_\sigma| = O(mr)$ and $\sum_{\sigma \in \Xi} |B(\sigma)| = O(n \log r)$.

For each cell $\sigma$ of the last cutting $\Xi_k$, we have $|\mathcal{C}_\sigma| \leq m/r$. Let $\hat{A}_\sigma$ denote the subset of points $p \in A$ such that $D_p$ has a bounding circle in $\mathcal{C}_\sigma$. We do not know whether distances between points of $\hat{A}_\sigma$ and points of $B(\sigma)$ are in $(\alpha, \beta]$ or not. If $|B(\sigma)| > n/r^2$, then we arbitrarily partition $B(\sigma)$ into subsets of size between $n/(2r^2)$ and $n/r^2$. We call these subsets *standard subsets* of $B(\sigma)$. Since $|B| = n$ and we have $O(r^2)$ cells in cutting $\Xi_k$, the number of standard subsets of all cells of $\Xi_k$ is $O(r^2)$. For each standard subset $\hat{B}(\sigma) \subseteq B(\sigma)$, we form a pair $(\hat{A}_\sigma, \hat{B}(\sigma))$ as an "unsolved" *subproblem*. Then we have $O(r^2)$ subproblems. Note that $|\hat{A}_\sigma| \leq m/r$ and $|\hat{B}(\sigma)| \leq n/r^2$. If we apply the same algorithm recursively on each subproblem, then we have the following recurrence relation (which holds for any $1 \leq r \leq m$):

$$T(m, n) = O(mr + n \log r) + O(r^2) \cdot T(\frac{m}{r}, \frac{n}{r^2}). \tag{1}$$

Note that if we use $T(m,n)$ to represent the total size of $A_t$ and $B_t$ of all complete bipartite graphs $A_t \times B_t$ in the subcollection of $\Gamma(A,B,\alpha,\beta)$ that have been produced as above, then we have the same recurrence as above. If $N(m,n)$ denotes the number of these graphs, then we have the following recurrence:

$$N(m,n) = O(r^2) + O(r^2) \cdot N(\frac{m}{r}, \frac{n}{r^2}).$$

We now solve the problem in a "dual" setting by switching the roles of $A$ and $B$, i.e., define annuli centered at points of $B$ and compute the hierarchical cutting for their bounding circles. Then, symmetrically we have the following recurrences (which holds for any $1 \le r \le n$):

$$T(m,n) = O(nr + m\log r) + O(r^2) \cdot T(\frac{m}{r^2}, \frac{n}{r}), \tag{2}$$

$$N(m,n) = O(r^2) + O(r^2) \cdot N(\frac{m}{r^2}, \frac{n}{r}).$$

By applying (2) to each subproblem of (1) using the same parameter $r$ and we can obtain the following recurrence:

$$T(m,n) = O(mr\log r + nr) + O(r^4) \cdot T(\frac{m}{r^3}, \frac{n}{r^3}).$$

Similarly, we have

$$N(m,n) = O(r^4) + O(r^4) \cdot N(\frac{m}{r^3}, \frac{n}{r^3}).$$

The above recurrences tell us that in $O(mr\log r + nr)$ time we can compute a collection of $O(r^4)$ edge-disjoint complete bipartite graphs $A_t \times B_t$ with $A_t \subseteq A$ and $B_t \subseteq B$ such that for any two points $a \in A_t$ and $b \in B_t$ their distance $\|ab\|$ lies in $(\alpha,\beta]$. Further, the size of all such $A_t$'s and $B_t$'s is bounded by $O(mr\log r + nr)$. We return the above collection as $\Gamma(A,B,\alpha,\beta)$ for the lemma.

In addition, we have also $O(r^4)$ graphs $A'_s \times B'_s$ with $A'_s \subseteq A$ and $B'_s \subseteq B$ corresponding to the unsolved subproblems $T(m/r^3, n/r^3)$ and we do not know whether $\|ab\| \in (\alpha,\beta]$ for points $a \in A'_s$ and $b \in B'_s$. We return the collection of all such graphs as $\Pi(A,B,\alpha,\beta)$ for the lemma. Hence, $|\Pi(A,B,\alpha,\beta)| = O(r^4)$, and $|A'_s| \le m/r^3$ and $|B'_s| \le n/r^3$ for each graph $A'_s \times B'_s$ in the collection. The number of pairs of points recorded in $\Pi(A,B,\alpha,\beta)$ is $O(|\Pi(A,B,\alpha,\beta)| \cdot m/r^3 \cdot n/r^3)$, which is $O(mn/r^2)$. This proves the lemma. ◀

Theorem 2 solves the complete BRS by running the algorithm of Lemma 1 recursively.

▶ **Theorem 2.** *We can compute in $O(m^{2/3}n^{2/3} \cdot 2^{O(\log^*(m+n))} + m\log n + n\log m)$ time a collection $\Gamma(A,B,\alpha,\beta) = \{A_t \times B_t \mid A_t \subseteq A, B_t \subseteq B\}$ of edge-disjoint complete bipartite graphs that satisfy the conditions of Problem 1 (with $\Pi(A,B,\alpha,\beta) = \emptyset$), with the following complexities: (1) $|\Gamma| = O(m^{2/3}n^{2/3}\cdot\log^*(m+n)+m+n)$; (2) $\sum_t |A_t|, \sum_t |B_t| = O(m^{2/3}n^{2/3} \cdot 2^{O(\log^*(m+n))} + m\log n + n\log m)$.*

**Proof.** To solve the complete BRS problem, the main idea is to apply the recurrence (2) recursively until the size of each subproblem becomes $O(1)$. We first consider the symmetric case where $m = n$. By setting $r = n^{1/3}/\log n$ and applying (2) with $m = n$, we obtain the following

$$T(n,n) = O(n^{4/3}) + O(n^{4/3}/\log^4 n) \cdot T(\log^3 n, \log^3 n). \tag{3}$$

Similarly, we have

$$N(n,n) = O(n^{4/3}/\log^4 n) + O(n^{4/3}/\log^4 n) \cdot N(\log^3 n, \log^3 n). \tag{4}$$

The recurrences solve to $T(n,n) = n^{4/3} \cdot 2^{O(\log^* n)}$ and $N(n,n) = O(n^{4/3} \cdot \log^* n)$. This means that in $n^{4/3} \cdot 2^{O(\log^* n)}$ time we can compute a collection $\Gamma(A,B,\alpha,\beta) = \{A_t \times B_t \mid A_t \subseteq A, B_t \subseteq B\}$ of $O(n^{4/3}\log^* n)$ edge-disjoint complete bipartite graphs, with $\sum_t |A_t|, \sum_t |B_t| = n^{4/3} \cdot 2^{O(\log^* n)}$, and it satisfies the conditions of Problem 1 with $\Pi(A,B,\alpha,\beta) = \emptyset$.

For the asymmetric case, i.e., $m \neq n$, it is solved by utilizing the above symmetric case result; the details can be found in the full paper. ◀

For comparison, Katz and Sharir [14] solved the complete BRS problem in $O((m^{2/3}n^{2/3} + m + n)\log m)$ time by producing $O(m^{2/3}n^{2/3} + m + n)$ complete bipartite graphs whose total vertex set size is $O((m^{2/3}n^{2/3} + m + n)\log m))$. Our result improves their runtime and vertex set size by almost a logarithmic factor with slightly more graphs produced.

## 3 Distance selection

In this section, we present our algorithm for the distance selection problem. Let $P$ be a set of $n$ points in the plane. Define $\mathcal{E}(P)$ as the set of distances of all pairs of points of $P$. Given an integer $1 \leq k \leq \binom{n}{2}$, the problem is to find the $k$-th smallest value in $\mathcal{E}(P)$, denoted by $\delta^*$.

Given any $\delta$, the *decision problem* is to determine whether $\delta \geq \delta^*$. Wang [18] recently gave an $O(n^{4/3})$ time algorithm that can compute the number of values of $\mathcal{E}(P)$ at most $\delta$, denoted by $k_\delta$. Observe that $\delta \geq \delta^*$ if and only if $k_\delta \geq k$. Thus, using Wang's algorithm [18], the decision problem can be solved in $O(n^{4/3})$ time. We should point out that the $O(n^{4/3}\log^2 n)$ time algorithm of Katz and Sharir [14] for computing $\delta^*$ utilizes a decision algorithm of $O(n^{4/3}\log n)$ time. However, even if we replace their decision algorithm by Wang's $O(n^{4/3})$ time algorithm, the runtime of the overall algorithm for computing $\delta^*$ is still $O(n^{4/3}\log^2 n)$ because other parts of the algorithm dominate the total time. To reduce the overall time to $O(n^{4/3}\log n)$, new techniques are needed, in addition to using the faster $O(n^{4/3})$ time decision algorithm. These new techniques include, for instance, Lemma 1 for the partial BRS problem, as will be seen below.

Before presenting the details of our algorithm, we first give the following lemma, which is critical to our algorithm and is obtained by using Lemma 1.

▶ **Lemma 3.** *Given an interval $(\alpha,\beta]$, Problem 1 with $A = P$ and $B = P$ can be solved in $O(n^{4/3})$ time by computing two collections $\Gamma(P,P,\alpha,\beta) = \{A_t \times B_t \mid A_t, B_t \subseteq P\}$ and $\Pi(P,P,\alpha,\beta) = \{A'_s \times B'_s \mid A'_s, B'_s \subseteq P\}$ with the following complexities: (1) $|\Gamma| = O(n^{4/3}/\log^4 \log n)$; (2) $\sum_t |A_t|, \sum_t |B_t| = O(n^{4/3})$; (3) $|\Pi| = O(n^{4/3}/\log^4 \log n)$; (4) $|A'_s|, |B'_s| = O(\log^3 \log n)$, for each $A'_s \times B'_s \in \Pi$.*

**Proof.** We first apply Lemma 1 with $A = P$, $B = P$, and $r = n^{1/3}/\log n$. This constructs a collection $\Gamma_1 = \{A_t \times B_t \mid A_t, B_t \subseteq P\}$ of $O(n^{4/3}/\log^4 n)$ edge-disjoint complete bipartite graphs in $O(n^{4/3})$ time. The total size of vertex sets of these graphs is $O(n^{4/3})$, i.e., $\sum_t |A_t|, \sum_t |B_t| = O(n^{4/3})$. We also have a collection $\Pi_1 = \{A'_s \times B'_s \mid A'_s, B'_s \subseteq P\}$ of $O(n^{4/3}/\log^4 n)$ edge-disjoint complete bipartite graphs that record uncertain point pairs, with $|A'_s|, |B'_s| = O(\log^3 n)$.

Hence, the number of uncertain pairs of points of $P$ (i.e., we do not know whether their distances are in $(\alpha,\beta]$) is $\sum_s |A'_s| \cdot |B'_s| = O(n^{4/3}\log^2 n)$. To further reduce this number, we apply Lemma 1 on every pair $(A'_s, B'_s)$ of $\Pi_1$. More specifically, for each pair $(A'_s, B'_s)$

of $\Pi_1$, we apply Lemma 1 with $A = A'_s$, $B = B'_s$, and $r = \log n / \log\log n$. This computes a collection $\Gamma_s$ of $O(\log^4 n / \log^4 \log n)$ edge-disjoint complete bipartite graphs in $O(\log^4 n)$ time; the total size of vertex sets of all graphs in $\Gamma_s$ is $O(\log^4 n)$. We also have a collection $\Pi_s$ of $O(\log^4 n / \log^4 \log n)$ edge-disjoint complete bipartite graphs. The size of each vertex set of each graph of $\Pi_s$ is bounded by $O(\log^3 \log n)$. The total time for Lemma 1 on all pairs $(A'_s, B'_s)$ of $\Pi_1$ as above is $O(n^{4/3})$. We return $\Gamma_1 \cup \bigcup_s \Gamma_s$ as collection $\Gamma$, and $\bigcup_s \Pi_s$ as collection $\Pi$ in the lemma statement. As such, the complexities in the lemma hold. ◀

In what follows, we describe our algorithm for computing $\delta^*$. Like Katz and Sharir's algorithm [14], our algorithm proceeds in stages. Initially, we have $I_0 = (0, +\infty]$. In each $j$-th stage, an interval $I_j = (\alpha_i, \beta_j]$ is computed from $I_{j-1}$ such that $I_j$ must contain $\delta^*$ and the number of values of $\mathcal{E}(P)$ in $I_j$ is a constant fraction of that in $I_{j-1}$. Specifically, we will prove that $|\mathcal{E}(P) \cap I_j| = O(n^2 \rho^j)$ holds for each $j$, for some constant $\rho < 1$. Once $|\mathcal{E}(P) \cap I_j|$ is no more than a threshold (to be given later; as will be seen later, this threshold is not constant, which is a main difference between our algorithm and Katz and Sharir's algorithm [14]), we will compute $\delta^*$ directly. In the following we discuss the $j$-th stage of the algorithm. We assume that we have an interval $I_{j-1} = (\alpha_{j-1}, \beta_{j-1}]$ containing $\delta^*$.

We first apply Lemma 3 with $(\alpha, \beta] = (\alpha_{j-1}, \beta_{j-1}]$. This is another major difference between our algorithm and Katz and Sharir's algorithm [14], where they solved the complete BRS problem, while we only solve a partial problem (this saves time by a logarithmic factor). Applying Lemma 3 produces a collection $\Gamma_{j-1} = \{A_t \times B_t \mid A_t, B_t \subseteq P\}$ of $O(n^{4/3} / \log^4 \log n)$ edge-disjoint complete bipartite graphs, with $\sum_t |A_t|, \sum_t |B_t| = O(n^{4/3})$, as well as another collection $\Pi_{j-1}$ of $O(n^{4/3} / \log^4 \log n)$ graphs. By Lemma 3 (3) and (4), the number of pairs of points of $P$ in $\Pi_{j-1}$ is $O(n^{4/3} \log^2 \log n)$.

If $\sum_t |A_t| \cdot |B_t| \leq n^{4/3} \log n$, which is our threshold, then this is the last stage of the algorithm and we compute $\delta^*$ directly by Lemma 4. Each edge of the graph in $\Gamma_{j-1} \cup \Pi_{j-1}$ connects two points of $P$; we say that the distance of the two points is *induced* by the edge.

▶ **Lemma 4.** *If $\sum_t |A_t| \cdot |B_t| \leq n^{4/3} \log n$, then $\delta^*$ can be computed in $O(n^{4/3} \log n)$ time.*

**Proof.** We first explicitly compute the set $S$ of distances induced from edges of all graphs of $\Gamma_{j-1}$ and $\Pi_{j-1}$. Since $\sum_t |A_t| \cdot |B_t| \leq n^{4/3} \log n$ and the number of edges of all graphs of $\Pi_{j-1}$ is $O(n^{4/3} \log^2 \log n)$, we have $|S| = O(n^{4/3} \log n)$ and $S$ can be computed in $O(n^{4/3} \log n)$ time by brute force. Then, we compute the number $k_{\alpha_{j-1}}$ of values of $\mathcal{E}(P)$ that are at most $\alpha_{j-1}$, which can be done in $O(n^{4/3})$ time [18]. Observe that $\delta^*$ is the $(k - k_{\alpha_{j-1}})$-th smallest value in $S$. Hence, using the linear time selection algorithm, we can find $\delta^*$ in $O(|S|)$ time, which is $O(n^{4/3} \log n)$. ◀

We now assume $\sum_t |A_t| \cdot |B_t| > n^{4/3} \log n$. The rest of the algorithm for the $j$-th iteration takes $O(n^{4/3})$ time. For each graph $A_t \times B_t \in \Gamma_{j-1}$, if $|A_t| < |B_t|$, then we switch the name of $A_t$ and $B_t$, i.e., $A_t$ now refers to $B_t$ and $B_t$ refers to the original $A_t$. Note that this does not change the solution of the partial BRS produced by Lemma 3 and it does not change the complexities of Lemma 3 either. This name change is only for ease of the exposition. Now we have $|A_t| \geq |B_t|$ for each graph $A_t \times B_t \in \Gamma_{j-1}$. Let $m_t = |A_t|$ and $n_t = |B_t|$.

We partition each $A_t$ into $g = \lfloor m_t / n_t \rfloor$ subsets $A_{t1}, A_{t2}, \ldots, A_{tg}$ so that each subset contains $n_t$ elements except that the last subset $A_{tg}$ contains at least $n_t$ but at most $2n_t - 1$ elements. Each pair $(A_{ti}, B_t)$, $1 \leq i \leq g$, can be viewed as a complete bipartite graph. As in [14], we construct a *$d$-regular LPS-expander graph $G_{ti}$* on the vertex set $A_{ti} \cup B_t$, for a

constant $d$ to be fixed later.[2] The expander $G_{ti}$ has $O(|A_{ti}|+|B_t|)$ edges and can be computed in $O(|A_{ti}|+|B_t|)$ time [14, 16]. Let $G_t$ be the union of all these expander graphs $G_{ti}$ over all $i = 1, 2, \ldots, g$. The construction of $G_t$ takes $\sum_{i=1}^{g} O(|A_{ti}| + |B_t|) = O(|A_t| + \lfloor \frac{m_t}{n_t} \rfloor \cdot |B_t|) = O(|A_t|)$ time. Hence, computing all graphs $\{G_t\}_t$ for all $O(n^{4/3}/\log^4 \log n)$ pairs $A_t \times B_t$ in $\Gamma_{j-1}$ takes $\sum_t O(|A_t|) = O(n^{4/3})$ time. The number of edges in $G_t$ is $O(|A_t| + |B_t|)$, and thus the number of edges in all graphs $\{G_t\}_t$ is $\sum_t O(|A_t| + |B_t|) = O(n^{4/3})$.

For each edge $(a, b)$ in graph $G_t$ that connects a point $a \in A_t$ and a point $b \in B_t$, we associate it with the interpoint distance $\|ab\|$. We compute all these distances for all graphs $\{G_t\}_t$ to form a set $S$. The size of $S$ is bounded by the number of edges in all graphs $\{G_t\}_t$, which is $O(n^{4/3})$. Note that all values of $S$ are in the interval $I_{j-1}$.

One way we could proceed from here is to find the largest value $\delta_1$ of $S$ with $\delta_1 < \delta^*$ and the smallest value $\delta_2$ with $\delta^* \leq \delta_2$, and then return $(\delta_1, \delta_2]$ as the interval $I_j$ and finish the $j$-th stage of the algorithm. Finding $\delta_1$ and $\delta_2$ could be done by binary search on $S$ using the linear time selection algorithm and the $O(n^{4/3})$ time decision algorithm. Then the runtime of this step would be $O(n^{4/3} \log n)$, resulting in a total of $O(n^{4/3} \log^2 n)$ time for the overall algorithm for computing $\delta^*$ since there are $O(\log n)$ stages. To improve the time, as in [14], we use the "Cole-like" technique to reduce the number of calls to the decision algorithm to $O(1)$ in each stage, as follows.

We assign a *weight* to each value of $S$. Note that since each graph $G_{ti} \in G_t$ is a $d$-regular LPS-expander, the degree of $G_{ti}$ is $d$ [14]. Hence, $G_{ti}$ has at most $(|A_{ti}| + |B_t|) \cdot d/2$ edges and thus it contributes at most $(|A_{ti}| + |B_t|) \cdot d/2$ values to $S$. We assign each distance induced from $G_{ti}$ a weight equal to $|A_{ti}| \cdot |B_t|/(|A_{ti}| + |B_t|)$. As such, the total weight of the values of $S$ is at most

$$\sum_{t,i} (|A_{ti}| + |B_t|) \cdot \frac{d}{2} \cdot \frac{|A_{ti}| \cdot |B_t|}{|A_{ti}| + |B_t|} = \frac{d}{2} \cdot \sum_{t,i} |A_{ti}| \cdot |B_t| = \frac{d}{2} \cdot m_{j-1},$$

where $m_{j-1} = \sum_t |A_t| \cdot |B_t|$. Recall that $m_{j-1} > n^{4/3} \log n$ and $|B_t| \leq |A_{ti}|$ in each $G_{ti}$. We can assume $n \geq 16$ so that $m_{j-1} \geq 16$. As such, we have the following bound for the weight of each value in $S$: $|A_{ti}| \cdot |B_t|/(|A_{ti}| + |B_t|) \leq |B_t| \leq \sqrt{|B_t| \cdot |A_{ti}|} \leq \sqrt{m_{j-1}} \leq m_{j-1}/4$.

We partition the values of $S$ into at most $2d$ intervals $\{I'_1, I'_2, ..., I'_h\}$, $1 \leq h \leq 2d$, such that the total weight of values in every interval is at least $m_{j-1}/4$ and but at most $m_{j-1}/2$. The partition can be done in $O(|S|)$ time, which is $O(n^{4/3})$, using the linear time selection algorithm. Then, we invoke the decision algorithm $\log(2d) = O(1)$ times to find the interval $I'_l$ that contains $\delta^*$, for some $1 \leq l \leq h$. We set $I_j = I'_l$. Since the decision algorithm is called $O(1)$ times, this step takes $O(n^{4/3})$ time. This finishes the $j$-th stage of the algorithm.

The following Lemma 5 shows that the number of values of $\mathcal{E}(P)$ in $I_j$ is a constant portion of that in $I_{j-1}$. This guarantees that the algorithm will finish in $O(\log n)$ stages since $|\mathcal{E}(P)| = O(n^2)$. As each stage runs in $O(n^{4/3})$ time (except that the last stage takes $O(n^{4/3} \log n)$ time), the total time of the algorithm is $O(n^{4/3} \log n)$.

▶ **Lemma 5.** *There exists a constant $\rho$ with $0 < \rho < 1$ such that the number of values of $\mathcal{E}(P)$ in $I_j$ is at most $\rho$ times the number of values of $\mathcal{E}(P)$ in $I_{j-1}$.*

[2] A good summary of definitions and properties of expanders can be found in Section 2 of [14]. Here it suffices for the reader to know the following property (which is needed in the proof of Lemma 5): If $X$ and $Y$ are two vertex subsets of a $d$-regular expander graph of $M$ vertices and there are fewer than $3M$ edges connecting points of $X$ and points of $Y$, then $|X| \cdot |Y| \leq 9M^2/d$.

**Proof.** Define $n_j$ (resp., $n_{j-1}$) as the number of values of $\mathcal{E}(P)$ in $I_j$ (resp., $I_{j-1}$). Our goal is to find a constant $\rho \in (0,1)$ so that $n_j \leq \rho \cdot n_{j-1}$ holds.

Recall that $m_{j-1}$ is the number of distances induced from the graphs of $\Gamma_{j-1}$. Define $m'_{j-1}$ as the number of distances induced from the graphs of $\Pi_{j-1}$. Define $q_j$ (resp., $q'_j$) as the number of interpoint distances of $\mathcal{E}(P) \cap I_j$ whose point pairs are recorded in $\Gamma_{j-1}$ (resp., $\Pi_{j-1}$). Note that all interpoint distances induced from graphs of $\Gamma_{j-1}$ are in $I_{j-1}$. Hence, $m_{j-1} \leq n_{j-1}$. By definition, $n_j = q_j + q'_j$ and $q'_j \leq m'_{j-1}$. By Lemma 3 (3) and (4), we have $m'_{j-1} = O(n^{4/3} \log^2 \log n)$.

We first make the following **claim:** there exists a constant $\gamma \in (0, 1/3)$ such that $q_j \leq \gamma \cdot m_{j-1}$. The proof of this claim is similar to the analysis in [14] and can be found in the full paper. Next, we prove the lemma by using this claim.

As this is not the last stage of the algorithm (since otherwise $\delta^*$ would have already been computed without producing interval $I_j$), it holds that $m_{j-1} > n^{4/3} \log n$. Since $m'_{j-1} = O(n^{4/3} \log^2 \log n)$, there exists a constant $c' \in (0, 1/3)$ such that $\frac{m'_{j-1}}{m_{j-1}} \leq c'$ when $n$ is sufficiently large. As $n_j = q_j + q'_j$, $q'_j \leq m'_{j-1}$, and $m_{j-1} \leq n_{j-1}$, we can obtain the following using the above claim:

$$n_j = q_j + q'_j \leq q_j + m'_{j-1} \leq \gamma \cdot m_{j-1} + c' \cdot m_{j-1} \leq (\gamma + c') \cdot m_{j-1} \leq (\gamma + c') \cdot n_{j-1}.$$

Set $\rho = \gamma + c'$. Since both $\gamma$ and $c'$ are in $(0, 1/3)$, we have $\rho \in (0, 2/3)$ and $n_j \leq \rho \cdot n_{j-1}$. This proves the lemma. ◀

We conclude with the following result. Note that once $\delta^*$ is computed, one can find a pair of points of $P$ whose distance is equal to $\delta^*$ in additional $O(n^{4/3})$ time [18].

▶ **Theorem 6.** *Given a set $P$ of $n$ points in the plane and an integer $1 \leq k \leq \binom{n}{2}$, the $k$-th smallest interpoint distance of $P$ can be computed in $O(n^{4/3} \log n)$ time.*

**Remark.** Our algorithm can be easily extended to the following *bipartite version* of the distance selection problem: Given a set $A$ of $m$ points and a set $B$ of $n$ points in the plane, and an integer $1 \leq k \leq mn$, compute the $k$-th smallest interpoint distance $\delta^*$ in the set $\{\|ab\| \mid a \in A, b \in B\}$. This problem can be solved in $O((m^{2/3}n^{2/3} + m \log n + n \log m) \log(m+n))$ time by extending our algorithm. More detailed discussions can be found in the full paper.

## 4 Two-sided discrete Fréchet distance with shortcuts

In this section, we show that our techniques in Section 3 can be used to solve the two-sided DFD problem. Let $A = \{a_1, a_2, ..., a_m\}$ and $B = \{b_1, b_2, ..., b_n\}$ be two sequences of points in the plane. Consider two frogs connected by an inelastic leash, initially placed at $a_1$ and $b_1$, respectively. Each frog is allowed to jump forward at most one step in one move, i.e., if the first frog is currently at $a_i$, then in the next move it can either jump to $a_{i+1}$ or stay at $a_i$. Note that frogs are not allowed to go backwards. The *discrete Fréchet distance* (or DFD for short) is defined as the minimum length of the inelastic leash that allows two frogs to reach their destinations, i.e., $a_m$ and $b_n$, respectively.

Because the Fréchet distance is very sensitive to outliers, to reduce the sensitivity, DFD with outliers have been proposed [4]. Specifically, if we allow the $A$-frog to jump from its current point to any of its succeeding points in each move but $B$-frog has to traverse all points in $B$ in order plus one restriction that only one frog is allowed to jump in each move (i.e., in each move one of the frogs must stay still), then this problem is called *one-sided discrete Fréchet distance with shortcuts* (or *one-sided DFD* for short), where the goal is to compute

the minimum length of the inelastic leash that allows two frogs to reach their destinations. If we allow both frogs to skip points in their sequences (but again with the restriction that only one frog is allowed to jump in each move), then problem is called *two-sided DFD*.

We focus on the two-sided DFD in this section while the one-sided version will be treated in the next section. Let $\delta^*$ denote the optimal objective value, i.e., the minimum length of the leash. Avraham, Filtser, Kaplan, Katz, and Sharir [4] presented an algorithm that can compute $\delta^*$ in $O((m^{2/3}n^{2/3}+m+n)\log^3(m+n))$ time. In what follows, we show that our techniques in Section 3 can improve their algorithm to $O((m^{2/3}n^{2/3}\cdot 2^{O(\log^*(m+n))}+m\log n+n\log m)\log(m+n))$ time, roughly a factor of $O(\log^2(m+n))$ faster.

To solve the problem, the authors of [4] first proposed an algorithm to solve the decision problem, i.e., given any $\delta$, decide whether $\delta^* \le \delta$; the algorithm runs in $O((m^{2/3}n^{2/3}+m+n)\log^2(m+n))$ time. Then, to compute $\delta^*$, the authors of [4] used the bipartite version of the distance selection algorithm from Katz and Sharir [14] for point sets $A$ and $B$ together with their decision algorithm to do binary search on the interpoint distances between points in $A$ and those in $B$, i.e., in each iteration, using the distance selection algorithm to find the $k$-th smallest distance $\delta_k$ for an appropriate $k$ and then call the decision algorithm on $\delta_k$ to decide which way to search. As both the distance selection algorithm [14] and the decision algorithm run in $O((m^{2/3}n^{2/3}+m+n)\log^2(m+n))$ time, computing $\delta^*$ takes $O((m^{2/3}n^{2/3}+m+n)\log^3(m+n))$ time.

The following lemma (whose proof is in the full paper) shows that the runtime of their decision algorithm [4] can be improved by a factor of roughly $O(\log^2(m+n))$, by using our result in Theorem 2 for the complete BRS problem.

▶ **Lemma 7.** *Given any $\delta$, we can decide whether the two-sided DFD $\delta^* \le \delta$ in $O(m^{2/3}n^{2/3}\cdot 2^{O(\log^*(m+n))}+m\log n+n\log m)$ time.*

**Improving the optimization algorithm for computing $\delta^*$.** With our new $O((m^{2/3}n^{2/3}+m\log n+n\log m)\log(m+n))$ time bipartite distance selection algorithm in Section 3 and the above faster decision algorithm, following the same binary search scheme as discussed above, $\delta^*$ can be computed in $O((m^{2/3}n^{2/3}+m\log n+n\log m)\log^2(m+n))$ time, a logarithmic factor improvement over the result of [4]. Notice that the time is dominated by the calls to the bipartite distance selection algorithm.

To further improve the algorithm, an observation is that we do not have to call the distance selection algorithm as an oracle and instead we can use that algorithm as a framework and replace the decision algorithm of the distance selection problem by the decision algorithm of the two-sided DFD problem. This will roughly reduce another logarithmic factor. The proof of the following theorem provides the details about this idea.

▶ **Theorem 8.** *Given two sequences of points $A=(a_1,a_2,...,a_m)$ and $B=(b_1,b_2,...,b_n)$ in the plane, the two-sided DFD problem can be solved in $O((m^{2/3}n^{2/3}\cdot 2^{O(\log^*(m+n))}+m\log n+n\log m)\log(m+n))$ time.*

**Proof.** Following our distance selection algorithm, we run in stages and each $j$-th stage will compute an interval $I_j$ that contains $\delta^*$. In the $j$-th stage, we first perform the partial BRS on point sets $A$ and $B$ with respect to interval $I_{j-1}$, in the same way as before. This produces a collection $\Gamma$ of $(m^{2/3}n^{2/3}/\log^4\log(m^2/n)+m^{2/3}n^{2/3}/\log^4\log(n^2/m)+m+n)$ edge-disjoint complete bipartite graphs that record some pairs of $A\times B$ whose interpoint distances are in $I_{j-1}$. The total size of vertex sets of all graphs in $\Gamma$ is $O(m^{2/3}n^{2/3}+m\log n+n\log m)$. In addition, we also have a collection $\Pi$ of complete bipartite graphs representing $O(m^{2/3}n^{2/3}\log^2\log(m+n))$ uncertain pairs of $A\times B$. The total runtime is $O(m^{2/3}n^{2/3}+m\log n+n\log m)$.

We next compute the number $n_\Gamma$ of distances induced from the graphs of $\Gamma$. If $n_\Gamma$ is larger than the threshold $\tau = (m^{2/3}n^{2/3} + m\log n + n\log m)\log(m+n)$, then we use the "Cole-like" technique to perform a binary search on the interpoint distances induced from the expander graphs that are built on the vertex sets of the graphs in $\Gamma$, which calls the decision algorithm $O(1)$ times. The runtime for this stage is $O(m^{2/3}n^{2/3} \cdot 2^{O(\log^*(m+n))} + m\log n + n\log m)$. If $n_\Gamma \leq \tau$, then we reach the last stage of the algorithm and we can compute $\delta^*$ as follows. We compute the interpoint distances induced from the graphs in $\Gamma$ and $\Pi$. The total number of such distances is $O((m^{2/3}n^{2/3} + m\log n + n\log m)\log(m+n))$. Using the decision algorithm and the linear time selection algorithm, a binary search on these interpoint distances is performed to compute $\delta^*$, which takes $O((m^{2/3}n^{2/3} \cdot 2^{O(\log^*(m+n))} + m\log n + n\log m)\log(m+n))$ time as the decision algorithm is called $O(\log(m+n))$ times. The algorithm finishes within $O(\log(m+n))$ stages by an analysis similar to Lemma 5 (indeed, the proof of Lemma 5 does not rely on which decision algorithm is used).

In summary, the total runtime for computing $\delta^*$ is bounded by $O((m^{2/3}n^{2/3} \cdot 2^{O(\log^*(m+n))} + m\log n + n\log m)\log(m+n))$. ◀

**A general (deterministic) algorithmic framework.** The algorithm of Theorem 8 can be made into a general algorithmic framework for solving geometric optimization problems involving interpoint distances in the plane. Specifically, suppose we have an optimization problem $\mathcal{P}$ whose optimal objective value $\delta^*$ is equal to $\|ab\|$ for a point $a \in A$ and a point $b \in B$, with $A$ as a set of $m$ points and $B$ as a set of $n$ points in the plane. The goal is to compute $\delta^*$. Suppose that we have a decision algorithm that can determine whether $\delta \geq \delta^*$ in $T_D$ time for any $\delta$. Then, we can compute $\delta^*$ by applying exactly the same algorithm of Theorem 8 except that we use the decision algorithm for $\mathcal{P}$ instead. The total time of the algorithm is $O((m^{2/3}n^{2/3} + m\log n + n\log m + T_D) \cdot \log(m+n))$. Note that in the case $T_D = o((m^{2/3}n^{2/3} + m\log n + n\log m)\log(m+n))$ this is faster than the traditional binary search approach by repeatedly invoking the distance selection algorithm.

▶ **Theorem 9.** *Given two sets $A$ and $B$ of $m$ and $n$ points respectively in the plane, any geometric optimization problem whose optimal objective value is equal to the distance between a point of $a \in A$ and a point of $b \in B$ can be solved in $O((m^{2/3}n^{2/3} + m\log n + n\log m + T_D) \cdot \log(m+n))$ time, where $T_D$ is the time for solving the decision version of the problem.*

## 5 One-sided discrete Fréchet distance with shortcuts

We consider the one-sided DFD problem defined in Section 4. Let $\delta^*$ denote the optimal objective value. Avraham, Filtser, Kaplan, Katz, and Sharir [4] proposed an a randomized algorithm of $O((m+n)^{6/5+\epsilon})$ expected time. We show that using our result in Lemma 1 for the partial BRS the runtime of their algorithm can be reduced to $O((m+n)^{6/5}\log^{8/5}(m+n))$.

Define $\mathcal{E}(A,B) = \{\|ab\| \mid a \in A, b \in B\}$. It is known that $\delta^* \in \mathcal{E}(A,B)$ [4]. The decision problem is to decide whether $\delta \geq \delta^*$ for any $\delta$. The authors [4] solved the decision problem in $O(m+n)$ (deterministic) time. To compute $\delta^*$, their algorithm has two main procedures.

The first main procedure computes an interval $(\alpha, \beta]$ that is guaranteed to contain $\delta^*$, and in addition, with high probability the interval contains at most $L$ values of $\mathcal{E}(A,B)$, given any $1 \leq L \leq mn$; the algorithm runs in $O((m+n)^{4/3+\epsilon}/L^{1/3} + (m+n)\log(m+n)\log\log(m+n))$ time, for any $\epsilon > 0$. More specifically, during the course of the algorithm, an interval $(\alpha, \beta]$ containing $\delta^*$ is maintained; initially $\alpha = 0$ and $\beta = \infty$. In each iteration, the algorithm first determines, through random sampling, whether the number of values of $\mathcal{E}(A,B)$ in $(\alpha, \beta]$ is at most $L$ with high probability. If so, the algorithm stops by returning the current interval

$(\alpha, \beta]$. Otherwise, a subset $R$ of $O(\log(m+n))$ values of $\mathcal{E}(A, B)$ is sampled which contains with high probability an approximate median (in the middle three quarters) among the values of $\mathcal{E}(A, B)$ in $(\alpha, \beta]$. A binary search guided by the decision algorithm is performed to narrow down the interval $(\alpha, \beta]$; the algorithm then proceeds with the next iteration. As such, after $O(\log(m+n))$ iterations, the algorithm eventually returns an interval $(\alpha, \beta]$ with the property discussed above.

The second main procedure is to find $\delta^*$ from $\mathcal{E}(A, B) \cap (\alpha, \beta]$. This is done by using a *bifurcation tree* technique (Lemma 4.4 [4]), whose runtime relies on $L'$, the true number of values of $\mathcal{E}(A, B)$ in $(\alpha, \beta]$. As it is possible that $L' > L$, if the algorithm detects that case happens, then the first main procedure will run one more round from scratch. As $L' < L$ holds with high probability, the expected number of rounds is $O(1)$. If $L' \leq L$, the runtime of the second main procedure is bounded by $O((m+n)L^{1/2}\log(m+n))$.

As such, the expected time of the algorithm is $O((m+n)^{4/3+\epsilon}/L^{1/3} + (m+n)\log(m+n)\log\log(m+n) + (m+n)L^{1/2}\log(m+n))$. Setting $L$ to $O((m+n)^{2/5+\epsilon})$ for another small $\epsilon > 0$, the time can be bounded by $O((m+n)^{6/5+\epsilon})$.

**Our improvement.** We can improve the runtime of the first main procedure by a factor of $O((m+n)^{\epsilon})$, which leads to the improvement of overall algorithm by a similar factor. To this end, by applying Lemma 1 with $r = (\frac{m+n}{L})^{1/3}$, we first have the following corollary, which improves Lemma 4.1 in [4] (which is needed in the first main procedure).

▶ **Corollary 10.** *Given a set $A$ of $m$ points and a set $B$ of $n$ points in the plane, an interval $(\alpha, \beta]$, and a parameter $1 \leq L \leq mn$, we can compute in $O((m+n)^{4/3}/L^{1/3} \cdot \log(\frac{m+n}{L}))$ time two collections $\Gamma(A, B, \alpha, \beta) = \{A_t \times B_t \mid A_t \subseteq A, B_t \subseteq B\}$ and $\Pi(A, B, \alpha, \beta) = \{A'_s \times B'_s \mid A'_s \subseteq A, B'_s \subseteq B\}$ of edge-disjoint complete bipartite graphs that satisfy the conditions of Problem 1, with the following complexities: (1) $|\Gamma| = O((\frac{m+n}{L})^{4/3})$; (2) $\sum_t |A_t|, \sum_t |B_t| = O((m+n)^{4/3}/L^{1/3} \cdot \log(\frac{m+n}{L}))$; (3) $|\Pi| = O((\frac{m+n}{L})^{4/3})$; (4) $|A'_s| = O(\frac{mL}{m+n})$ and $|B'_s| = O(\frac{nL}{m+n})$ for each $A'_s \times B'_s \in \Pi$; (5) the number of pairs of points recorded in $\Pi$ is $O((m+n)^{4/3}L^{2/3})$.*

Replacing Lemma 4.1 in [4] by our results in Corollary 10 and following the rest of the algorithm in [4] leads to an algorithm to compute $\delta^*$ in $O((m+n)^{6/5}\log^2(m+n))$ time. More details can be found in the full paper, which makes the discussion in the context of a more general algorithmic framework (indeed, a recent result of Katz and Sharir [15] already gave such a framework; here we improve their result by a factor of $O((m+n)^{\epsilon})$ due to Corollary 10). As discussed in Section 1, another immediate application of the framework is the reverse shortest path problem in unit-disk graphs [20].

## References


1 Pankaj K. Agarwal, Boris Aronov, Micha Sharir, and Subhash Suri. Selecting distances in the plane. *Algorithmica*, 9(5):495–514, 1993.

2 Pankaj K. Agarwal, Rinat B. Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43:429–449, 2014.

3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995.

4 Rinat B. Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Transactions on Algorithms*, 11(4):Article No. 29, 2015.

5 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 645–654, 2009.

6 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the 30th Annual Symposium on Computational Geometry (SoCG)*, pages 367–376, 2014.

7 Timothy M. Chan. On enumerating and selecting distances. *International Journal of Computational Geometry and Application*, 11:291–304, 2001.

8 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

9 Timothy M. Chan and Da Wei Zheng. Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 190–210, 2022. Full version with new results available at `arXiv:2111.03744`.

10 Bernard Chazelle. New techniques for computing order statistics in Euclidean space. In *Proceedings of the 1st Annual Symposium on Computational Geometry (SoCG)*, pages 125–134, 1985.

11 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete and Computational Geometry*, 9(2):145–158, 1993.

12 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

13 Michael T. Goodrich. Geometric partitioning made easier, even in parallel. In *Proceedings of the 9th Annual Symposium on Computational Geometry (SoCG)*, pages 73–82, 1993.

14 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.

15 Matthew J. Katz and Micha Sharir. Efficient algorithms for optimization problems involving semi-algebraic range searching. *arXiv*, 2021. `arXiv:2111.02052`.

16 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 240–246, 1986.

17 Jiří Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters*, 39:183–187, 1991.

18 Haitao Wang. Unit-disk range searching and applications. In *Proceedings of the 18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 32:1–32:17, 2022.

19 Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020.

20 Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. *Journal of Computational Geometry*, 14(1):14–47, 2023.

21 Andrew Chi-Chih Yao. On constructing minimum spanning trees in $k$-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

# Maximum Coverage in Random-Arrival Streams

**Rowan Warneke**
School of Computing and Information Systems, The University of Melbourne, Australia

**Farhana Choudhury**
School of Computing and Information Systems, The University of Melbourne, Australia

**Anthony Wirth**
School of Computing and Information Systems, The University of Melbourne, Australia

## Abstract

Given a collection of $m$ sets, each a subset of a universe $\{1, \ldots, n\}$, *maximum coverage* is the problem of choosing $k$ sets whose union has the largest cardinality. A simple greedy algorithm achieves an approximation factor of $1 - 1/e \approx 0.632$, which is the best possible polynomial-time approximation unless P = NP.

In the streaming setting, information about the input is revealed gradually, in an online fashion. In the set-streaming model, each set is listed contiguously in the stream. In the more general edge-streaming model, the stream is composed of set-element pairs, denoting membership. The overall goal in the streaming setting is to design algorithms that use sublinear space in the size of the input. An interesting line of research is to design algorithms with space complexity polylogarithmic in the size of the input (i.e., polylogarithmic in both $n$ and $m$); we call such algorithms low-space. In the set-streaming model, it is known that $1/2$ is the best possible low-space approximation. In the edge-streaming model, no low-space algorithm can achieve a nontrivial approximation factor.

We study the problem under the assumption that the order in which the stream arrives is chosen uniformly at random. Our main results are as follows.

- In the random-arrival set-streaming model, we give two new algorithms to show that low space is sufficient to break the $1/2$ barrier. The first achieves an approximation factor of $1/2 + c_1$ using $\tilde{O}(k^2)$ space, where $c_1 > 0$ is a small constant and $\tilde{O}(\cdot)$ notation suppresses polylogarithmic factors; the second achieves a factor of $1 - 1/e - \varepsilon - o(1)$ using $\tilde{O}(k^2\varepsilon^{-3})$ space, where the $o(1)$ term is a function of $k$. This is essentially the optimal bound, as breaking the $1 - 1/e$ barrier is known to require high space.
- In the random-arrival edge-streaming model, we show for all fixed $\alpha > 0$ and $\delta > 0$, any algorithm that $\alpha$-approximates maximum coverage with probability at least 0.9 in the random-arrival edge-streaming model requires $\Omega(m^{1-\delta})$ space (i.e., high space), even for the special case of $k = 1$.

**2012 ACM Subject Classification** Theory of computation → Random order and robust communication complexity; Theory of computation → Sketching and sampling

**Keywords and phrases** Maximum Coverage, Streaming Algorithm, Random Arrival, Greedy Algorithm, Communication Complexity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.102

**Acknowledgements** Thanks to Andrew McGregor for suggesting this as a possible research direction.

## 1 Introduction

*Maximum coverage* is a classic NP-hard problem in computer science with a range of applications, including facility allocation [16], information retrieval [2], and content recommendation [18]. We are given a collection of $m$ sets, $\mathcal{F}$, each a subset of a universe $[n] \coloneqq \{1, \ldots, n\}$, and a positive integer, $k$. The goal is to choose $k$ sets from $\mathcal{F}$ such that the cardinality of the union of the chosen sets – known as their *coverage* – is maximised. A standard greedy algorithm achieves a $(1 - 1/e)$-approximation and unless P = NP, this is the best approximation possible in polynomial time [7].


The greedy algorithm performs well in practice, but requires access to the entire input during its operation, which scales poorly for massive data sets. In the last 15 years, there has been an increasing focus on *streaming algorithms* for maximum coverage [18, 15, 6, 11]. These algorithms process the input piece by piece and use substantially less space than would be required to simply read the entire input into memory and run a classical algorithm, which requires $O(mn)$ space in general. In the *set-streaming* model, each set arrives contiguously in the stream: all the elements in one set are listed, then all the elements in the next set, and so on. In the more general *edge-streaming* model, each entity in the stream is a set-element pair, so the full description of a set might be spread across the stream.

Most existing work on streaming algorithms for maximum coverage has focused on the *arbitrary-arrival* model, in which the stream may arrive in any order. In particular, the stream may arrive in a *worst-case* order, so any theoretical guarantees made about an algorithm in this model must be made for all possible stream orderings. In practice, however, it is often reasonable to assume that the data stream is randomly ordered;[1] in the *random-arrival* model, all possible stream orderings are considered equally likely, and theoretical guarantees about an algorithm in this model need only be made as an *average* over these orderings. The random-arrival model has been studied for many important problems, including quantile estimation [10], maximum matching [13, 4], and submodular maximisation [17, 1], often revealing improved space-accuracy trade-offs.

In this paper, we study maximum coverage in the random-arrival model. In the *random-arrival set-streaming* model,[2] the order in which the sets appear in the stream is uniformly random, but we assume nothing about the order in which the elements in each set arrive. By contrast, in the *random-arrival edge-streaming* model, the edges arrive in a uniformly random order, so large sets are more likely to be represented early in the stream. For each model, our goal is to determine whether the random-arrival assumption permits a better approximation factor than is possible in the corresponding arbitrary-arrival model, when very low space is available (i.e., polylogarithmic in both $n$ and $m$). We answer this question in the affirmative for random-arrival set-streaming: we present two low-space algorithms, each achieving an approximation factor better than $1/2$ (the best possible approximation in the arbitrary-arrival model). For random-arrival edge-streaming, we demonstrate a near-tight space lower bound, hence answering in the negative: nearly linear in $m$ space is required to achieve a nontrivial approximation.

## 1.1 Related work

Table 1 summarises relevant past work on streaming algorithms for maximum coverage. Note that any algorithm that guarantees an approximation factor in the arbitrary-arrival model trivially also guarantees this approximation factor in the random-arrival model.

The multi-pass variant of streaming maximum coverage has also been studied, in which a small number of passes over the stream are allowed [15, 12]. In this paper, however, we focus exclusively on single-pass algorithms.

Unless otherwise specified, the following results apply to the arbitrary-arrival model.

---

[1] See Guha and McGregor [10] §1.1 for a discussion on justifications for this assumption.

[2] Note that we consider both "random-arrival" and "set-streaming" to be part of the model specification. When we talk about *the* random-arrival model, we mean both the random-arrival set-streaming and random-arrival edge-streaming models.

**Set-streaming.** Saha and Getoor [18], who introduced the set-streaming model, gave a 1/4-approximation algorithm called SOPS,[3] which uses $\tilde{O}(nk)$ space and explicitly returns the $k$ chosen sets. Yu and Yuan [20] studied the problem under the more relaxed ID-reporting output specification, in which only the *IDs* of the $k$ chosen sets must be returned by the algorithm.[4] They gave an algorithm, GOPS, which achieves an approximation factor of approximately 0.3 and uses $\tilde{O}(n)$ space, where $\tilde{O}(\cdot)$ notation suppresses dependence on polylogarithmic factors. Further progress came indirectly from Badanidiyuru et al. [5], who gave a $(1/2-\varepsilon)$-approximation algorithm for the related submodular maximisation problem. A careful adaptation to maximum coverage uses $\tilde{O}(n\varepsilon^{-1})$ space [15].

More recently, McGregor and Vu [15] developed the first set-streaming maximum coverage algorithms that use sublinear-in-$n$ space. Using *subsampling*, which effectively discards much of the universe, $[n]$, they substantially reduce the space consumption of their algorithms with minimal loss in solution quality. Of the four new algorithms they developed, the fourth, which we call MV-4 after the authors, is the most relevant to our work, as it is both low-space and single-pass. MV-4 achieves an approximation factor of $1/2-\varepsilon$ using just $\tilde{O}(k\varepsilon^{-3})$ space. The algorithm is explained in detail in Section 2.1. The other two MV algorithms that are also single-pass are also included in Table 1 (although note that they are not low-space).

McGregor and Vu [15] also proved the first nontrivial space lower bound for set-streaming maximum coverage. They showed that achieving an approximation factor better than $1-1/e$ requires $\Omega(mk^{-2})$ space in the arbitrary-arrival model and $\Omega(mk^{-3})$ space in the random-arrival model. Feldman et al. [8] extended this result to approximation factors better than $1/2$ for the arbitrary-arrival model.[5] Note that all lower bounds discussed in this work apply to the problem of simply *estimating* the optimal coverage.

The results of McGregor and Vu [15] and Feldman et al. [8] imply that the best possible approximation factor that can be achieved using low space in the arbitrary-arrival model is $1/2$, where by "low space" we mean space polylogarithmic in both $n$ and $m$.[6] In the random-arrival model, however, there is a knowledge gap for approximation factors between $1/2$ and $1-1/e$: can some low-space algorithm achieve an approximation factor in this range, or is high space always required?

**Edge-streaming.** Bateni et al. [6] gave the first edge-streaming maximum coverage algorithm, which achieves an approximation factor of $1-1/e-\varepsilon$ using $\tilde{O}_\varepsilon(m)$ space, where the subscript suppresses dependence on $\varepsilon$ (which was not analysed by the authors). Indyk and Vakilian [11] improved this result by showing that the optimal space bound is $\tilde{\Theta}(\alpha^2 m)$, where $\alpha > 0$ is the approximation factor. This result implies that achieving a nontrivial (i.e., nonzero) approximation factor using low space is impossible in the arbitrary-arrival edge-streaming model.

[3] The algorithm was given this name in a later work [20].

[4] IDs appear explicitly in the edge-streaming model as part of the set-element pairs. In the set-streaming model, we assume for convenience that the ID of each set is listed directly before the elements in the set, but this is a largely unimportant implementation detail.

[5] The authors actually studied the more general submodular maximisation problem. However, the submodular function constructed for the hardness instance is also a coverage function, so the result applies to maximum coverage as well.

[6] Other reasonable definitions exist, but we choose to focus on $n$ and $m$ since these are the parameters that can cause a maximum coverage problem instance to require a huge amount of space to store, which is the motivation for using streaming algorithms in the first place.

**Table 1** A summary of known results and our new results for the space complexity of single-pass streaming approximation algorithms for maximum coverage. Each of the parameters $\alpha, \delta, \varepsilon$ is positive, and $c_1$ is a small positive constant.

| Stream | Name | Arrival | Approximation | Space | Ref. |
|---|---|---|---|---|---|
| Set | SOPS | Arbitrary | $1/4$ | $O(nk)$ | [18] |
| | GOPS | Arbitrary | $\sim 0.3$ | $\tilde{O}(n)$ | [20] |
| | MCSS | Arbitrary | $1/2 - \varepsilon$ | $\tilde{O}(n\varepsilon^{-1})$ | [5, 15] |
| | MV-1 | Arbitrary | $1 - 1/e - \varepsilon$ | $\tilde{O}(m\varepsilon^{-2})$ | [15] |
| | MV-3 | Arbitrary | $1 - \varepsilon$ | $\tilde{O}(m\varepsilon^{-2} \cdot \min(k, \varepsilon^{-1}))$ | [15] |
| | MV-4 | Arbitrary | $1/2 - \varepsilon$ | $\tilde{O}(k\varepsilon^{-3})$ | [15] |
| | – | Arbitrary | $> 1 - 1/e$ | $\Omega(mk^{-2})$ | [15] |
| | – | Arbitrary | $> 1/2$ | $\Omega(mk^{-3})$ | [8] |
| | – | Random | $> 1 - 1/e$ | $\Omega(mk^{-3})$ | [15] |
| | GS-SALSA | Random | $1/2 + c_1$ | $\tilde{O}(k^2)$ | **Here** |
| | GS-SMC$^+$ | Random | $1 - 1/e - \varepsilon - o(1)$ | $\tilde{O}(k^2\varepsilon^{-3})$ | **Here** |
| Edge | – | Arbitrary | $1 - 1/e - \varepsilon$ | $\tilde{O}_\varepsilon(m)$ | [6] |
| | – | Arbitrary | $\alpha > 0$ | $\tilde{\Theta}(\alpha^2 m)$ | [11] |
| | – | Random | Any fixed $\alpha > 0$ | $\Omega(m^{1-\delta})$ | **Here** |

## 1.2 Our contributions

The results of this paper are included in Table 1.

**Set-streaming.** In the arbitrary-arrival model, the best approximation factor that can be achieved using low space is $1/2$. We present two low-space algorithms that break the $1/2$ barrier in the random-arrival model.

- The first algorithm, GS-SALSA, uses $\tilde{O}(k^2)$ space and achieves an approximation factor of $1/2 + c_1$ in-expectation, where $c_1 > 0$ is a small absolute constant.
- The second algorithm, GS-SMC$^+$, uses $\tilde{O}(k^2\varepsilon^{-3})$ space and achieves an approximation factor of $1 - 1/e - \varepsilon - o(1)$ in-expectation, where the $o(1)$ term is a function of $k$. For large $k$, this is essentially the optimal low-space approximation.[7]

Both algorithms are based on existing state-of-the-art random-arrival algorithms for the related *submodular maximisation* problem [17, 1]. Our main contribution is a generalisation of the subsampling technique introduced by McGregor and Vu [15] in the design of MV-4, which we apply to these existing submodular maximisation algorithms to achieve our results.

**Edge-streaming.** Indyk and Vakilian [11] showed that $\tilde{\Theta}(\alpha^2 m)$ space is necessary and sufficient for $\alpha$-approximating maximum coverage in the arbitrary-arrival model. An immediate corollary is that for all *fixed* $\alpha > 0$, the problem requires $\Omega(m)$ space. We prove an almost matching hardness result for the random-arrival case.

[7] Throughout this work, we typically do *not* consider the approximation factor of an algorithm (or hardness result) as a function of $k$, and instead consider the *worst-case* value of $k$ (just as we take a worst-case collection of sets $\mathcal{F}$). We make an exception for GS-SMC$^+$ due to the quality of its approximation for large values of $k$. Note that the hardness results of McGregor and Vu [15] and Feldman et al. [8] apply as $k$ approaches infinity, so the algorithm really is near-optimal for large $k$.

▶ **Theorem 1.** *For all fixed $\alpha > 0$ and $\delta > 0$, any algorithm that $\alpha$-approximates maximum coverage with probability at least 0.9 in the random-arrival edge-streaming model requires $\Omega(m^{1-\delta})$ space, even for the special case of $k = 1$.*

Our result implies that unlike set-streaming, edge-streaming maximum coverage is *not* made easier by the random-arrival assumption, in the sense that the assumption does not increase the best possible low-space approximation. Our proof of Theorem 1 is a careful combination of ideas from the proof for the arbitrary-arrival case by Indyk and Vakilian [11] with ideas from Andoni et al. [3], who proved a random-arrival space lower bound for the *frequency moment estimation* problem.

## 2 Beating 1/2 for Set-Streaming

In this section, we show that the 1/2 barrier can be broken for random-arrival set-streaming maximum coverage using low space. Our main contribution is a generalised version of the subsampling technique introduced by McGregor and Vu [15], which allows us to replace the core subroutine of MV-4 with *any* set-streaming maximum coverage algorithm. In particular, replacing the subroutine with state-of-the-art streaming algorithms for the related *submodular maximisation* problem yields two new maximum coverage algorithms with approximation factors better than 1/2 in the random-arrival model. The first, GS-SALSA, achieves an approximation factor of $1/2 + c_1$ for a small constant $c_1 > 0$. The second, GS-SMC$^+$, achieves an approximation factor of $1 - 1/e - \varepsilon - o(1)$, where the $o(1)$ term is a function of $k$. For large $k$, this is essentially the optimal bound.

### 2.1 Preliminaries: Design of MV-4

We start by providing a high-level overview of the MV-4 algorithm designed by McGregor and Vu [15], which achieves an approximation factor of $1/2 - \varepsilon$ using $\tilde{O}(k\varepsilon^{-3})$ space in the arbitrary-arrival model, which is essentially the optimal low-space approximation.

At the heart of MV-4 is a subroutine, $\mathcal{A}$, which achieves an approximation factor of $1/2 - \varepsilon$ using high space. This seeming contradiction is avoided by only providing a small sample of the input to $\mathcal{A}$; MV-4 is equipped with a binary hash function $h : [n] \to \{0, 1\}$, and as each element $e$ arrives in the stream, it is passed to the subroutine $\mathcal{A}$ only if $h(e) = 1$. The subroutine is therefore run on a *subsampled* problem instance $\mathcal{I}'$ over a smaller universe $[n]' = \{e \in [n] : h(e) = 1\}$, with subsampled sets $S'_i = S_i \cap [n]'$ for each $S_i \in \mathcal{F}$. An optimal solution to $\mathcal{I}'$ may not correspond to an optimal solution to $\mathcal{I}$, but when the hash function is chosen appropriately, a good solution to $\mathcal{I}'$ corresponds to a nearly-as-good solution to $\mathcal{I}$ with high probability. The hash function $h$ is chosen such that $\mathbb{P}[h(e) = 1] = p$ for all $e \in [n]$. When the subsampling rate $p$ is small, the space consumption of $\mathcal{A}$ is substantially reduced, and the overall algorithm becomes low-space.

The subsampling rate $p$ is set in terms of the optimal coverage, OPT. Of course, we do not know OPT in advance! To get around this, MV-4 runs parallel instantiations of $\mathcal{A}$, each corresponding to a different guess $v$ for OPT. Each instantiation $\mathcal{A}_v$ therefore has its own subsampling rate $p_v$, hash function $h_v$, and is run on a separate subsampled instance $\mathcal{I}'_v$. This introduces two new problems. Firstly, at the end of the stream, how do we know which instantiation corresponds to the correct guess? Rather than solve this problem directly, MV-4 maintains an $F_0$-*sketch*[8] of the set of elements covered (in the *unsubsampled* universe, $[n]$)

[8] Given a stream of items, an $F_0$-sketch is a small data structure capable of closely estimating the number

by the solution found by each instantiation. At the end of the stream, the solution with the highest estimated coverage is taken. Jaud et al. [12] gave a simpler method, which uses the *subsampled* coverage achieved by each instantiation to decide which solution to return.

The second problem introduced by the use of parallel instantiations concerns the space consumption of instantiations corresponding to bad guesses for OPT. If a guess $v$ is much smaller than OPT, the subsampling rate $p_v$ is set too high, the subsampled universe is too large, and so $\mathcal{A}_v$ uses too much space. To address this, MV-4 keeps track of the space consumption of each instantiation and terminates the instantiation if it uses too much space.

## 2.2 Generalised subsampling

In this section, we present a generalised version of McGregor and Vu's [15] subsampling technique. This allows the subroutine $\mathcal{A}$ to be replaced by *any* set-streaming maximum coverage algorithm $\mathcal{B}$. In Section 2.3, we apply our generalised subsampling technique to maximum coverage algorithms derived from state-of-the-art algorithms for submodular maximisation. The space consumption of these algorithms is proportional to $d = \max_{S_i \in \mathcal{F}} |S_i|$, the maximum set size of the problem instance. Under this condition, generalised subsampling substantially reduces the space consumption of $\mathcal{B}$. Our main result is as follows.

▶ **Theorem 2** (Generalised subsampling). *Suppose $\mathcal{B}$ achieves an approximation factor of $\alpha$ in-expectation*[9] *for set-streaming maximum coverage using $O(ds)$ space, where $d$ is the maximum set size. There exists an algorithm, called GS($\mathcal{B}$), which, given $\varepsilon > 0$, achieves an approximation factor of $\alpha - \varepsilon$ in-expectation and uses $\tilde{O}(k\varepsilon^{-2}s)$ space.*

An important aspect of Theorem 2 is that no assumptions are made about the design of $\mathcal{B}$. This presents a challenge when we try to replace $\mathcal{A}$ with $\mathcal{B}$ in MV-4, as the operation of MV-4 relies on an important property of $\mathcal{A}$. In particular, $\mathcal{A}$ is *threshold-based*: as each set arrives in the stream, the algorithm decides irrevocably whether to include the set in the solution, and maintains a set $C$ of the elements covered by the chosen sets. The decision to include an incoming set is based on whether the additional coverage provided by the set exceeds some threshold. This property is important to the design of MV-4 for two reasons. Firstly, it allows MV-4 to maintain a sketch of the elements covered by each instantiation of $\mathcal{A}$ in the unsubsampled universe, since after each set arrives, $\mathcal{A}$ can inform MV-4 as to whether it chose the set.[10] Secondly, the space consumption of $\mathcal{A}$ is directly proportional to the size of $C$, so this value may be used to trigger the termination of instantiations that use too much space.

We do not want to assume that $\mathcal{B}$ is threshold-based – indeed, we want to assume nothing at all about the design $\mathcal{B}$ – so we use an alternative idea. We abandon guessing OPT, and instead guess $d$, the maximum set size.[11] Our approach requires that some guess $w$ satisfies $d/2 \leq w \leq d$, so we make guesses in powers of 2. The problem of choosing which

---

of distinct items in the stream. The details are unimportant for our purposes.

[9] There is nothing special about in-expectation guarantees, and the theorem could just as easily be proved for with-high-probability guarantees. We focus on in-expectation results since this is the type of guarantee made about the state-of-the-art submodular maximisation algorithms that we apply our results to in Section 2.3.

[10] The alternative approach used by Jaud et al. [12] works quite differently, but also exploits the threshold-based architecture of $\mathcal{A}$ by using the size of $C$ for each instantiation to decide which solution to return.

[11] This is similar to an idea appearing in Badanidiyuru et al. [5], who studied streaming algorithms for submodular maximisation. Their algorithm, SIEVE-STREAMING, runs parallel instantiations corresponding to guesses for the maximum value of the given submodular function over any one item from the stream.

**Algorithm 1** The generalised subsampling algorithm GS($\mathcal{B}$).

```
1:  W ← {2^i : i ∈ ℕ, 2^i ≤ n}                          ▷ Guesses for d
2:  λ ← ⌊2k log(emε^{-1})⌋                              ▷ Hash function independence parameter
3:  for w ∈ W do
4:      Initialise B_w, an instantiation of B
5:      p_w ← min{1, 3kε^{-2} log(emε^{-1}) w^{-1}}     ▷ Subsampling rate
6:      Sample h_w ∈ H_{p_w,λ} uniformly at random      ▷ Choose hash function
7:      active_w ← true                                 ▷ Kill switch indicator
8:  for i = 1, ..., m do                                ▷ Iterate over each set in the stream
9:      for e ∈ S_i do                                  ▷ Iterate over each element in the set
10:         for w ∈ W do
11:             if d*_w > 2p_w w(1 + ε) then
12:                 active_w ← false                    ▷ Terminate B_w
13:             if active_w and h_w(e) = 1 then
14:                 Supply e to the stream of B_w
15: Let I_w ⊂ [m] be the solution returned by B_w
16: w_c ← min{w ∈ W : d*/2 ≤ w ≤ d*}                     ▷ At this point, d* = d.
17: return I_{w_c}
```

solution to return becomes trivial, since we can easily keep track of the maximum observed set size and choose the appropriate instantiation at the end of the stream (Line 16). We also keep track of the maximum observed set size for each *subsampled* problem instance, and use this value to trigger the termination of bad instantiations (Line 11).

Our generalised subsampling algorithm GS($\mathcal{B}$) is formalised by Algorithm 1. The variable $d^*$ keeps track of the running maximum set size observed so far in the stream. For example, $d^* = 0$ at the start of the stream, and $d^* = d$ at the end of the stream. Similarly, $d^*_w$ keeps track of the maximum set size observed for the subsampled problem instance $\mathcal{I}'_w$ corresponding to the guess $w$ for $d$. We omit from Algorithm 1 the straightforward steps required to keep track of these variables. The set $\mathcal{H}_{p_w,\lambda}$ appearing on Line 6 represents a $\lambda$-wise independent family of hash functions with the property that for all $e \in [n]$, $\mathbb{P}[h(e) = 1] = p_w$. A hash function $h$ from this family may be sampled, stored, and evaluated using $O(\lambda)$ space [15]. In the remainder of this section, we show that GS($\mathcal{B}$) achieves the approximation factor and space consumption guaranteed by Theorem 2.

**Approximation factor.** GS($\mathcal{B}$) returns the solution found by $\mathcal{B}_{w_c}$, where $w_c$ is the smallest guess for $d$ satisfying $d/2 \leq w_c \leq d$ (see Line 16), so we focus on this instantiation of $\mathcal{B}$. In particular, $S'_i$ is taken to mean the version of $S_i$ subsampled using the hash function $h_{w_c}$, and the $w_c$ subscript is often suppressed (e.g., $p = p_{w_c}$ and $\mathcal{I}' = \mathcal{I}'_{w_c}$).

Since each element $e \in [n]$ is subsampled with probability $p$, we expect that for any given choice of sets $S_1, \ldots, S_l$, the subsampled coverage of these sets will be approximately $p$ times their unsubsampled coverage. The following result formalises this intuition.

▶ **Lemma 3.** *With probability at least $1 - \varepsilon$, for all collections of up to $k$ sets $S_1, \ldots, S_l \in \mathcal{F}$, $|S'_1 \cup \cdots \cup S'_l| = p \cdot |S_1 \cup \cdots \cup S_l| \pm p\varepsilon d$.*

This result is similar to Lemma 8 from McGregor and Vu [15], with a few minor changes. Most notably, the probability of success is changed from a term involving $m$ to one involving $\varepsilon$, which is important for the purpose of making an in-expectation guarantee.

Our proof more closely resembles the proof of Corollary 5 from Jaud et al. [12], who showed that the independence factor $\lambda$ can be decreased substantially from its original definition in McGregor and Vu [15]. As in Jaud et al. [12], we require the following concentration bound for our proof.

▶ **Theorem 4** (Schmidt et al. [19]). *Let $X_1, \ldots, X_n$ be $\lambda$-wise independent binary random variables. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbb{E}[X]$. If $\lambda \leq \lfloor \min(\gamma, \gamma^2)\mu e^{-1/3} \rfloor$, then*

$$\mathbb{P}\left[|X - \mu| \geq \gamma\mu\right] \leq e^{-\lfloor \lambda/2 \rfloor}.$$

**Proof of Lemma 3.** Fix any collection of sets $S_1, \ldots, S_l \in \mathcal{F}$ with $1 \leq l \leq k$. Let $D = |S_1 \cup \cdots \cup S_l|$ be the unsubsampled coverage of the collection and let $D' = |S'_1 \cup \cdots \cup S'_l|$ be the subsampled coverage. Let $X_e = 1$ if $e \in S_1 \cup \cdots \cup S_l$ and $h(e) = 1$, and let $X_e = 0$ otherwise. Then $D' = \sum_{i=1}^{n} X_i$ and $\mu \coloneqq \mathbb{E}[D'] = pD$. The $X_i$ are $\lambda$-wise independent by the choice of $h$, where $\lambda = \lfloor 2k \log(em\varepsilon^{-1}) \rfloor$. Define $\gamma = \varepsilon d/D$ so that $\gamma\mu = p\varepsilon d$. Before applying Theorem 4, we verify the necessary condition on the independence factor:

$$\begin{aligned}\left\lfloor \min(\gamma, \gamma^2)\mu e^{-1/3} \right\rfloor &= \left\lfloor \min(1, \gamma)\gamma\mu e^{-1/3} \right\rfloor \geq \left\lfloor \varepsilon \cdot p\varepsilon d \cdot \frac{2}{3} \right\rfloor \\ &= \left\lfloor \varepsilon^2 \cdot 3k\varepsilon^{-2} \log(em\varepsilon^{-1}) w^{-1} \cdot d \cdot \frac{2}{3} \right\rfloor \geq \left\lfloor 2k \log(em\varepsilon^{-1}) \right\rfloor = \lambda,\end{aligned}$$

where we make use of the fact that $\gamma \geq \varepsilon$ (since $d \geq D$), $e^{-1/3} \geq 2/3$, and $w \leq d$. Therefore, we have

$$\begin{aligned}\mathbb{P}\left[|D' - \mu| \geq p\varepsilon d\right] = \mathbb{P}\left[|D' - \mu| \geq \gamma\mu\right] &\leq \exp\left(-\left\lfloor \frac{\lfloor 2k \log(em\varepsilon^{-1}) \rfloor}{2} \right\rfloor\right) \\ &\leq \exp\left(-\left\lfloor k \log(em\varepsilon^{-1}) \right\rfloor\right) \leq \exp\left(-k \log(em\varepsilon^{-1}) + 1\right) \\ &\leq (em\varepsilon^{-1})^{-k} \cdot e = e^{-k+1} m^{-k} \varepsilon^k \leq m^{-k}\varepsilon.\end{aligned}$$

The total number of collections is $\sum_{i=1}^{k} \binom{m}{i} \leq \sum_{i=1}^{k} \frac{m^k}{k} = m^k$, so taking a union bound, the probability that $|D' - \mu| \geq p\varepsilon d$ for some collection is at most $m^k m^{-k}\varepsilon = \varepsilon$. ◀

Using Lemma 3, we can show that a good solution to the subsampled instance $\mathcal{I}'$ corresponds to a nearly-as-good solution to $\mathcal{I}$. This result is analogous to Corollary 9 from McGregor and Vu [15].

▶ **Corollary 5.** *Let $\mathrm{OPT}'$ be the optimal coverage for the subsampled problem instance $\mathcal{I}'$. If a choice of $k$ sets $S_1, \ldots, S_k$ satisfies $|S'_1 \cup \cdots \cup S'_k| \geq \beta \cdot \mathrm{OPT}'$, then with probability at least $1 - \varepsilon$, $|S_1 \cup \cdots \cup S_k| \geq (\beta - 2\varepsilon) \cdot \mathrm{OPT}$.*

**Proof.** Let $O_1, \ldots, O_k$ be an optimal solution to the unsubsampled problem instance $\mathcal{I}$. We have

$$\mathrm{OPT}' \geq |O'_1 \cup \cdots \cup O'_k| \geq p \cdot |O_1 \cup \cdots \cup O_k| - p\varepsilon d \geq p(1 - \varepsilon) \cdot \mathrm{OPT},$$

where the second inequality follows from Lemma 3. Now let $S_1, \ldots, S_k$ be a collection of sets satisfying $|S'_1 \cup \cdots \cup S'_k| \geq \beta \cdot \mathrm{OPT}$. Applying Lemma 3 once again, we have

$$\begin{aligned}|S_1 \cup \ldots \cup S_k| &\geq \frac{1}{p} \cdot |S'_1 \cup \ldots \cup S'_k| - \varepsilon d \geq \frac{1}{p} \cdot \beta \cdot \mathrm{OPT}' - \varepsilon \cdot \mathrm{OPT} \\ &\geq \frac{\beta}{p} \cdot p(1 - \varepsilon) \cdot \mathrm{OPT} - \varepsilon \cdot \mathrm{OPT} \geq (\beta - 2\varepsilon) \cdot \mathrm{OPT}.\end{aligned}$$

◀

Now $\mathcal{B}$ achieves an approximation factor of $\alpha$ in-expectation on the subsampled instance $\mathcal{I}'$. Therefore, by Corollary 5, with probability at least $1-\varepsilon$, $\mathcal{B}$ achieves an approximation factor of $\alpha - 2\varepsilon$ in-expectation. The unconditional expected approximation factor is therefore at least $(1-\varepsilon)(\alpha-2\varepsilon) \geq (\alpha - 3\varepsilon)$. Dividing $\varepsilon$ by 3 at the start (omitted from Algorithm 1 for brevity) corrects this to $\alpha - \varepsilon$ without changing the asymptotic space complexity.

Of course, the above reasoning is only valid if, in the at-least-$(1-\varepsilon)$-probability event that all the approximations in Lemma 3 hold, the instantiation $\mathcal{B}_{w_c}$ is not terminated.[12] Let $i'_{\max} = \arg\max_i |S'_i|$ be the index of the largest set in the subsampled instance. We have

$$d^*_{w_c} \leq d_{w_c} = \left|S'_{i'_{\max}}\right| \overset{*}{\leq} p_{w_c} \left|S_{i'_{\max}}\right| + p_{w_c}\varepsilon d \leq p_{w_c} d(1+\varepsilon) \leq 2p_{w_c} w_c(1+\varepsilon),$$

where the starred inequality follows from an application of Lemma 3 on the singleton collection containing only $S_{i'_{\max}}$. Therefore, the instantiation $\mathcal{B}_{w_c}$ is not terminated.

**Space consumption.** We now consider an arbitrary instantiation $\mathcal{B}_w$, where $w$ is not necessarily the correct guess for $d$. Unlike $n$, $m$, and $k$, we do not assume that a maximum coverage streaming algorithm is provided with the maximum set size $d$ at the start of the stream. Furthermore, at any point during the stream, it could be that the largest set has already been observed (i.e., $d^* = d$). Therefore, since $\mathcal{B}_w$ uses $O(d_w s)$ space, it must also use $O(d^*_w s)$ space. Now as long as $\mathcal{B}_w$ is not terminated, by Line 11, we have $d^*_w \leq 2p_w w(1+\varepsilon)$, so the space consumption of $\mathcal{B}_w$ is

$$O(d^*_w s) = O(p_w w(1+\varepsilon)s) = O(k\varepsilon^{-2}\log(em\varepsilon^{-1})w^{-1}ws) = \tilde{O}(k\varepsilon^{-2}s).$$

Each instantiation also requires the storage of the hash function $h_w$, but this takes just $O(\lambda) = \tilde{O}(k)$ space. There are $|W| = O(\log n)$ instantiations, so the total space complexity of $\mathrm{GS}(\mathcal{B})$ is $\tilde{O}(k\varepsilon^{-2}s)$. This completes the proof of Theorem 2.

## 2.3 Beating 1/2 via submodular maximisation

Submodular maximisation is a heavily studied problem in combinatorial optimisation that may be viewed as a generalisation of maximum coverage. A function $f : 2^V \to \mathbb{R}$ over a ground set $V$ is said to be *submodular* if we have

$$f(e \mid X) \geq f(e \mid Y) \quad \text{for all } X \subseteq Y \subseteq V \text{ and } e \in V \setminus Y,$$

where $f(e \mid X) = f(X \cup \{e\}) - f(X)$ is the marginal gain of $e$ given $X$. Submodular maximisation is the problem of choosing $A \subseteq V$ such that $f(A)$ is maximised. We consider a popular variant of the problem in which a cardinality constraint $|A| \leq k$ is applied and $f$ is assumed to be non-negative and monotone.

Submodular maximisation has received a lot of attention in the streaming model [5, 17, 1]. In this model, the items of $V$ arrive one at a time in the stream, and we assume access to an oracle for $f$, which may be queried in $O(1)$ time. In the arbitrary-arrival model, it is known that $\Omega(mk^{-3})$ space is required to achieve an approximation factor above $1/2$, where $m = |V|$ is the length of the stream [8].[13] In the random-arrival model, however, the $1/2$ barrier has been broken. This was first achieved by Norouzi-Fard et al. [17], who gave a $\tilde{O}(k)$-space

[12] Note that the inequality in Corollary 5 always holds in this event.

[13] This is the same result that states that $\Omega(mk^{-3})$ space is required to achieve approximation factors better than $1/2$ for arbitrary-arrival set-streaming maximum coverage (cf. Table 1).

algorithm, called SALSA, which achieves an approximation factor of $1/2 + c_0$ in-expectation for a very small constant $c_0 > 0$. More recently, Agrawal et al. [1] gave an algorithm which achieves an approximation factor of $1 - 1/e - \varepsilon - o(1)$ in-expectation, where the $o(1)$ term is a function of $k$. The algorithm, which we call SMC after its authors, uses $\tilde{O}_\varepsilon(k)$ space, where a complicated exponential dependence on $\varepsilon$ is suppressed. Liu et al. [14] improved the space complexity to $\tilde{O}(k\varepsilon^{-1})$ while maintaining the approximation factor; we call this improved algorithm SMC$^+$.

Submodular maximisation generalises maximum coverage: given a maximum coverage instance $\mathcal{I} = (\mathcal{F}, k)$, we simply set $V = \mathcal{F}$ and define $f$ to be the coverage function $f(X) = |\cup_{S_i \in X} S_i|$. It is not hard to see that $f$ is non-negative, monotone, and submodular. In the study of maximum coverage, however, we do *not* assume oracle access for evaluating coverage, so it must be evaluated explicitly. Therefore, if we use a submodular maximisation streaming algorithm for set-streaming maximum coverage, entire sets must be retained in memory, leading to an increase in space consumption proportional to $d$, the size of the largest set. Applying Theorem 2, however, we have the following key result.

▶ **Corollary 6.** *Suppose $\mathcal{B}$ achieves an approximation factor of $\alpha$ in-expectation for streaming submodular maximisation using $O(s)$ space. There exists an algorithm for set-streaming maximum coverage which, given $\varepsilon > 0$, achieves an approximation factor of $\alpha - \varepsilon$ in-expectation and uses $\tilde{O}(k\varepsilon^{-2}s)$ space.*

Applying this result to SALSA with $\varepsilon = c_1 \coloneqq c_0/2$ yields GS-SALSA, which achieves an approximation factor of $1/2 + c_0 - \varepsilon = 1/2 + c_1$ in-expectation using $\tilde{O}(k\varepsilon^{-2} \cdot k\varepsilon^{-1}) = \tilde{O}(k^2)$ space (note that $\varepsilon$ in this case is constant). Applying the result to SMC$^+$ yields GS-SMC$^+$, which uses $\tilde{O}(k^2\varepsilon^{-3})$ space and achieves an approximation factor of $1 - 1/e - \varepsilon - o(1) - \varepsilon$ in-expectation. Dividing $\varepsilon$ by 2 returns the approximation factor to $1 - 1/e - \varepsilon - o(1)$ with no change to the space complexity. Recall that achieving an approximation factor above $1 - 1/e$ requires high space [15], so for large $k$, this is essentially the optimal approximation. The $o(1)$ term decreases quite slowly, however, so for small $k$, GS-SALSA may be preferable.

## 3 Hardness Result for Edge-Streaming

In this section, we prove Theorem 1, which essentially says that no low-space algorithm can achieve a nontrivial approximation in the random-arrival edge-streaming model.

Our proof is a reduction from the heavily studied *$t$-party set disjointness problem*, DISJ$_t$. In this problem, $t$ players are each given a set $S_i \subseteq [N]$, where the $i^{\text{th}}$ player knows only $S_i$. Either all sets are pairwise disjoint (a YES instance), or are pairwise disjoint except for an element that is common to all $t$ sets (a NO instance). Gronemeier [9] showed that any protocol that solves this problem with probability 2/3 requires $\Omega(N/t)$ bits of communication, even if the players may use public randomness. The hardness instance for this problem is such that we may assume that $|S_1| = \ldots = |S_t| = cN/t$ for an arbitrarily small constant $c > 0$.

Our proof combines ideas from two existing approaches, both of which are also reductions from DISJ$_t$. The first is a result due to Andoni et al. [3], who proved a near-tight space lower bound for the problem of *frequency moment estimation* on random-arrival streams. Given a stream $\langle a_1, \ldots, a_l \rangle$, where each $a_i \in [n]$, the $k^{\text{th}}$ frequency moment is defined as $F_k \coloneqq \sum_{i \in [n]} f_i^k$, where $f_i = |\{j : a_j = i\}|$ is the number of times $i$ appears in the stream. Andoni et al. [3] studied the problem of 0.5-estimating this quantity, which is known to require $\tilde{\Theta}(n^{1-2/k})$ space in the arbitrary-arrival model. They showed that $\Omega(n^{1-2.5/k}/\log n)$ space is necessary in the random-arrival model, almost matching the upper bound. In their

proof, a stream of integers is constructed such that we have $F_k = l$ for a YES instance and $F_k \geq 2l$ for a NO instance. Our proof is very similar, but we include all the details for completeness. The main difference in our proof is that we need to construct a stream of *edges*, rather than a stream of integers. We use an idea from Indyk and Vakilian [11], who proved that $\Omega(\alpha^2 m)$ space is required to $\alpha$-approximate maximum coverage in the arbitrary-arrival edge-streaming model. Their proof involves the construction of a stream of edges such that the maximum coverage is 1 for a YES instance and $1/\alpha$ for a NO instance. We make a small change to this construction to ensure that the stream is in random order.

We start with two slightly reparameterised preliminary results from Andoni et al. [3]. Throughout this section, we assume that $\alpha$ and $\delta$ have been fixed.

▶ **Lemma 7.** *Let $\mathcal{W} = \{I_1, \ldots, I_t\}$ be $t = l^{2\delta/5}$ random intervals from*

$$\text{Cycle}_{l,w} \coloneqq \{[i-1 \pmod l + 1, \ldots, w+i-2 \pmod l + 1] : i \in [l]\} ,$$

*where $w = c_1 l^{1-3\delta/5}$ and $c_1 > 0$ is a sufficiently small absolute constant. With probability at least 0.99,*

1. *$I_{i_1} \cap I_{i_2} \cap I_{i_3} = \emptyset$ for any $i_1 < i_2 < i_3$.*
2. *$|\{(i_1, i_2) : i_1 < i_2,\ I_{i_1} \cap I_{i_2} \neq \emptyset\}| \leq \sqrt{t}$.*

The set $\text{Cycle}_{l,w}$ is simply the set of size-$w$ intervals from $[l]$, including those that "wrap around" from $l$ to 1. For example, $\text{Cycle}_{4,3} = \{[1,2,3],[2,3,4],[3,4,1],[4,1,2]\}$.

▶ **Lemma 8.** *Consider a uniformly random subset $S \subseteq [l]$ of size $t = l^{2\delta/5}$. For some sufficiently small absolute constant $c_2 > 0$, with probability at least 0.99, for each $j_1, j_2 \in S$, $|j_2 - j_1| \geq c_2 l^{1-4\delta/5}$.*

We also require the following result.

▶ **Lemma 9.** *Let $X$ be the number of unique values produced by $\lceil t/4 \rceil$ independent, uniformly random draws from $[t]$ (with replacement). For sufficiently large $t$, $\mathbb{P}(X \geq t/6) \geq 0.99$.*

**Proof.** Let $X_i = \mathbf{1}(i \text{ is drawn at least once})$, then $X = \sum_{i=1}^{t} X_i$. We have

$$\mu \coloneqq \mathbb{E}[X] = t\mathbb{E}[X_1] = t\left(1 - (1-1/t)^{\lceil t/4 \rceil}\right) \geq t\left(1 - e^{-1/4}\right) \geq t/5.$$

By a Chernoff bound for negatively correlated Boolean random variables where we set $\beta = 6/5$, we have

$$\mathbb{P}\left(X < \frac{t}{6}\right) = \mathbb{P}\left(X < \frac{t/5}{\beta}\right) \leq \mathbb{P}\left(X < \frac{\mu}{\beta}\right) \leq \left(\frac{e^{1/\beta - 1}}{\beta^\beta}\right)^\mu = c_3^\mu \leq c_3^{t/5},$$

where $c_3$ is an absolute constant less than 1. By setting $t \geq 5 \log_{c_3}(0.01) \approx 11.9$, we get $\mathbb{P}(X < t/6) \leq 0.01$ and the result follows. ◀

We are now ready to prove Theorem 1. Let $\mathcal{S} = \{S_1, \ldots, S_t\}$ be an instance of $\text{DISJ}_t$ where $t = l^{2\delta/5}$, $N = l^{1-\delta/5}$, and for all $i \in [t]$, we have $|S_i| = c_1 N/t = c_1 l^{1-3\delta/5} =: w$, where $c_1$ is as in Lemma 7. Any protocol that solves $\mathcal{S}$ with probability at least 2/3 requires $\Omega(N/t) = \Omega(l^{1-3\delta/5})$ bits of communication.

Let $\mathcal{A}$ be an $s$-space algorithm for random-arrival edge-streaming maximum coverage that achieves an approximation factor of $\alpha$ with probability at least 0.9 on streams containing $l$ edges. We describe a protocol that uses $\mathcal{A}$ to solve $\mathcal{S}$ using $O(ts)$ bits of communication, and therefore deduce that $s = \Omega(l^{1-3\delta/5}/t) = \Omega(l^{1-\delta}) = \Omega(m^{1-\delta})$ (note that $m \leq l$, since $l$ edges cannot possibly specify the contents of more than $m$ non-empty sets).

Using public randomness, the players pick:

1. Intervals $\mathcal{W} = \{I_1 = [a_1, b_1], \ldots, I_t = [a_t, b_t]\}$ from $\text{Cycle}_{l,w}$, ordered such that $b_i \le b_j$ if $i \le j$. The intervals are chosen independently with replacement.
2. A permutation $\sigma$ of $[2l]$.
3. A binary string $r$ of length $t^2/c_2$, where $c_2$ is as in Lemma 8.

Each interval in $\mathcal{W}$ has size $|I_i| = w$. If $b_1 < w$, then at least one set "wraps around" from $l$ to 1. In this case, terminate with failure. Each interval has probability $(w-1)/l$ of wrapping around, so by a simple union bound, $\mathbb{P}[b_1 < w] \le t(w-1)/l \le c_1 l^{-\delta/5}$. For large enough $l$, this value is at most 0.01. Also, if any $j \in [n]$ appears in three or more distinct intervals in $\mathcal{W}$, or if more than $\sqrt{t}$ pairs of intervals intersect, the protocol terminates with failure. By Lemma 7, this too occurs with probability at most 0.01.

The players construct, as described below, a length-$l$ stream $U = \langle u_1, \ldots, u_l \rangle$ representing a maximum coverage problem instance $\mathcal{I} = (\mathcal{F} = \{H_1, \ldots, H_m\}, k = 1)$ in the edge-streaming model, where each $H_i \subseteq [t]$. Each edge is a set-element pair $u_j = (y_j, e_j)$ indicating that $e_j \in H_{y_j}$. The players take turns generating edges and feeding them to the stream of the algorithm $\mathcal{A}$. When it is no longer a player's turn to generate the next edge, the player sends the entire state of $\mathcal{A}$ to the appropriate player, which requires $O(s)$ bits of communication.

**Constructing the stream.** The construction is very similar to the one presented by Andoni et al. [3]. First, each player $i$ randomly orders their set $S_i$ to produce a string $s^i$ of length $w$. Now consider $u_j$, the $j^{\text{th}}$ element in the stream $U$. The player chosen to construct $u_j$ depends on the number of intervals in $\mathcal{W}$ that contain $j$.

0. If $j$ appears in none of the intervals, $u_j$ is determined by player $i$, where $I_i = [a_i, b_i]$ is the interval with the smallest $a_i$ such that $a_i > j$. If no such interval exists (which may occur towards the end of the stream) then $u_j$ is determined by player $t$.
1. If $j$ appears in just one interval $I_i$, $u_j$ is determined by player $i$.
2. If $j$ appears in two intervals $I_i$ and $I_{i+1}$, we have a "clash" between players $i$ and $i+1$. We partition $[l]$ into $t^2/c_2$ equally sized intervals, which we call *zones*. We say that zone $u$ is a *0-zone* if $r_u = 0$, and a *1-zone* otherwise. If $j$ is in a 0-zone, $u_j$ is determined by player $i$. Otherwise, $u_j$ is determined by player $i+1$. Each zone has length $w_2 \coloneqq c_2 l^{1-4\delta/5}$.

Note that $j$ never appears in more than two intervals, since this results in termination of the protocol. Figure 1 shows an example of how players might be chosen to construct the stream.

At the end of their interval, each player sends the state of $\mathcal{A}$ to the next player. Also, when two players' intervals overlap, they may need to send the state of $\mathcal{A}$ back and forth several times (e.g., in Figure 1, players 2 and 3 need to pass the state to each other three times). Assuming the protocol does not terminate, there are at most $\sqrt{t}$ clashes, and each clash results in at most $w/w_2 = O(\sqrt{t})$ messages being sent. The total number of messages is $O(t + \sqrt{t} \cdot \sqrt{t}) = O(t)$, so the communication complexity of the protocol is $O(ts)$.

Now suppose player $i$ has been chosen to construct the edge $u_j = (y_j, e_j)$. The set ID $y_j$ is chosen in exactly the same way as the integer $a_j$ in Andoni et al. [3]: set $y_j = \sigma(x_j)$, where the definition of $x_j$ differs for each of the three cases.

0. Set $x_j = l + j$ to be a *padding value*.
1. Set $x_j = s^i_q$ to be a *standard value* with probability 1/2, where $j$ is the $q^{\text{th}}$ element of $I_i$. Otherwise, set $x_j = l + j$ to be a padding value.
2. Set $x_j = s^i_q$ to be a standard value, where $j$ is the $q^{\text{th}}$ element of $I_i$.

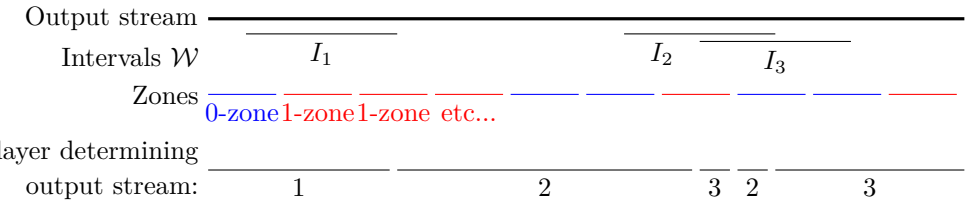



**Figure 1** An example allocation of stream construction responsibility for a simple $\text{DISJ}_3$ instance in which there is a single overlap between two intervals ($I_2$ and $I_3$). Note that this is a non-terminating instance.

**Table 2** A portion of an example reduction for a $\text{DISJ}_t$ NO instance with multiply occurring element 3. Column $j$ denotes the index of the stream, while the strings $s^2$ and $s^3$ (which contain the contents of the sets $S_2$ and $S_3$) have been positioned in place of the intervals $I_2 = \{11, \ldots, 14\}$ and $I_3 = \{13, \ldots, 16\}$ for clarity. The "Player" column denotes the player who determines the edge $u_j = (\sigma(x_j), p_j)$. In Case 1 (e.g., $j = 11$), there are two different equally likely values for $x_j$; both values are shown. We assume that $l = 20$.

| $j$ | $s^2$ | $s^3$ | Zone | Player | Case | $x_j$ |
|---|---|---|---|---|---|---|
| ⋮ | | | ⋮ | ⋮ | ⋮ | ⋮ |
| 10 | | | 0 | 2 | 0 | $l + j = 30$ |
| 11 | 4 | | 0 | 2 | 1 | $s^2_1 = 4$ or $l + j = 31$ |
| 12 | 5 | | 1 | 2 | 1 | $s^2_2 = 5$ or $l + j = 32$ |
| 13 | 3 | 8 | 1 | 3 | 2 | $s^3_1 = 8$ |
| 14 | 1 | 2 | 0 | 2 | 2 | $s^2_4 = 1$ |
| 15 | | 3 | 0 | 3 | 1 | $s^3_3 = 3$ or $l + j = 35$ |
| 16 | | 6 | 0 | 3 | 1 | $s^3_4 = 6$ or $l + j = 36$ |
| 17 | | | 0 | 4 | 0 | $l + j = 37$ |
| ⋮ | | | ⋮ | ⋮ | ⋮ | ⋮ |

Table 2 shows an example of how the $x_j$ are defined for a portion of the stream that includes two overlapping intervals. Observe that each occurrence of an element $e$ in a set $S_i$ has probability 1/2 of appearing as a standard value $x_j = e$ for some $j$.[14]

It remains to choose the element $e_j$. In their proof for the arbitrary-arrival case, Indyk and Vakilian [11] set $e_j = i$ and insert the edge $(x_j, e_j)$ into the stream, where $x_j$ is always a standard value (the intervals are contiguous sections of the stream). The result is that each $H_e \in \mathcal{F}$ contains the IDs of the players holding the element $e$, so the maximum coverage is either 1 (in a YES instance) or $t$ (in a NO instance). This idea does not work for our purposes, since if we set $e_j = i$, consecutive edges inserted by the same player will always have the same $e_j$. Instead, we set $e_j = p_j$, where $p_j \in [t]$ is a *randomly sampled* player ID.

By the same argument given by Andoni et al. [3], the stream is in (nearly[15]) random order. The only difference is the presence of $p_j$ in each of our edges. However, as the $p_j$ are randomly and independently sampled from $[t]$, they have no effect on whether the stream is randomly ordered.

[14] In Case 1, this probability comes from the random choice between a standard value and a padding value. In Case 2, this probability comes from the random choice of zone due to the binary string $r$.

[15] Certain orderings are impossible due to the termination with failure conditions. Since termination happens with probability at most 0.02, however, the biggest detrimental effect that this can possibly have on the performance of $\mathcal{A}$ is a decrease of 0.02 in the probability of successfully achieving the approximation factor.

**Using the output of $\mathcal{A}$.** What does the output of $\mathcal{A}$ tell us about the $\text{DISJ}_t$ instance $\mathcal{S}$? The following claims are made assuming that the protocol does not terminate with failure.

▷ Claim 10. If $\mathcal{S}$ is a YES instance, the optimal coverage of $\mathcal{I}$ is 1.

Proof. First, observe that the $x_j$ are all distinct, since all padding values are distinct from each other, all standard values are distinct from each other (as $\mathcal{S}$ is a YES instance), and no standard value can ever equal a padding value, as $s^i_q \leq N < l < l + j$. Therefore, since $\sigma$ is a permutation, the $y_j$ are also all distinct, so each edge in the stream specifies a different set ID. Every set is therefore singleton, so the optimal coverage is 1. ◁

▷ Claim 11. If $\mathcal{S}$ is a NO instance, then with probability at least 0.97, the optimal coverage of $\mathcal{I}$ is at least $t/6$.

Proof. There is some $e \in [N]$ such that $e \in S_i$ for all $i \in [t]$. For each occurrence of $e$ in a set $S_i$, there is some position in the stream where $e$ appears as an edge $(\sigma(e), p_j)$ with probability $1/2$. Let $V$ be this set of positions (e.g., in Table 2, the multiply occurring element is $e = 3$, and the set $V$ includes 13 and 15). Since the intervals $\mathcal{W}$ are chosen randomly and the strings $s^i$ are randomly ordered, $V$ is a uniformly random subset of $[l]$, so by Lemma 8, with probability at least 0.99, no two elements of $V$ fall within $c_2 l^{1-4\delta/5}$ of each other. This is precisely $w_2$, the size of each zone, so no two elements fall within the same zone. Each occurrence of $e$ therefore appears in the stream with probability $1/2$ *independently*, so the number of occurrences that appear has a Binomial distribution $q \sim B(t, 1/2)$. When $t$ is sufficiently large,[16] $q \geq t/4$ with probability at least 0.99.

Now consider these $q$ appearances, $(\sigma(e), p_1), \ldots, (\sigma(e), p_q)$. Assuming that we indeed have $q \geq t/4$, by Lemma 9, the number of distinct values among $p_1, \ldots, p_q$ is at least $t/6$, so choosing $H_{\sigma(e)}$ yields a coverage of at least $t/6$. By a union bound on the three sources of error (Lemma 8, variation in the Binomial distribution, and Lemma 9), this occurs with probability at least 0.97. ◁

By taking $l$ large enough, we can ensure that $t/6 \geq 1/\alpha$, in which case $\mathcal{A}$ is powerful enough to distinguish between YES and NO instances. Tallying up the various sources of error, $\mathcal{A}$ succeeds with probability at least 0.9, we avoid early termination with probability at least 0.98, and (in the case of a NO instance) Claim 11 holds with probability at least 0.97. Thus, by a union bound, the protocol succeeds with probability at least $1 - 0.1 - 0.02 - 0.03 = 0.85 \geq 2/3$. This completes the proof of Theorem 1.

## References


1. Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular Secretary Problem with Shortlists. In *10th ITCS*, pages 1:1–1:19, 2019.
2. Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering for fast approximate document retrieval. *ACM Transactions on Information Systems*, 33(3):1–35, 2015.
3. Alexandr Andoni, Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. Better bounds for frequency moments in random-order streams, 2008. `arXiv:0808.2222`.
4. Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *48th ICALP*, page 19, 2021.


[16] Note that we can make $t$ arbitrarily large by taking $l$ to be sufficiently large.

5 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *20th ACM SIGKDD*, pages 671–680, 2014.

6 MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. Almost optimal streaming algorithms for coverage problems. In *29th ACM SPAA*, pages 13–23, 2017.

7 Uriel Feige. A threshold of ln $n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

8 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *52nd ACM STOC*, pages 1363–1374, 2020.

9 André Gronemeier. Asymptotically optimal lower bounds on the NIH-multi-party information complexity of the AND-function and disjointness. In *26th STACS*, pages 505–516, 2009.

10 Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.

11 Piotr Indyk and Ali Vakilian. Tight trade-offs for the maximum $k$-coverage problem in the general streaming model. In *38th ACM PODS*, pages 200–217, 2019.

12 Stephen Jaud, Anthony Wirth, and Farhana Choudhury. Maximum coverage in sublinear space, faster, 2023. `arXiv:2302.06137`.

13 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *15th APPROX*, pages 231–242, 2012.

14 Paul Liu, Aviad Rubinstein, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. In *34th NeurIPS*, pages 6491–6502, 2021.

15 Andrew McGregor and Hoa T Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63:1595–1619, 2019.

16 Nimrod Megiddo, Eitan Zemel, and S Louis Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods*, 4(2):253–261, 1983.

17 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *35th ICML*, pages 3829–3838, 2018.

18 Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *9th SDM*, pages 697–708, 2009.

19 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

20 Huiwen Yu and Dayu Yuan. Set coverage problems in a one-pass data stream. In *13th SDM*, pages 758–766, 2013.

# Efficient Block Approximate Matrix Multiplication

**Chuhan Yang**
Division of Engineering, New York University Abu Dhabi, UAE
Tandon School of Engineering, New York University, NY, USA

**Christopher Musco**
Tandon School of Engineering, New York University, NY, USA

## Abstract

Randomized matrix algorithms have had significant recent impact on numerical linear algebra. One especially powerful class of methods are algorithms for approximate matrix multiplication based on sampling. Such methods typically sample individual matrix rows and columns using carefully chosen importance sampling probabilities. However, due to practical considerations like memory locality and the preservation of matrix structure, it is often preferable to sample contiguous blocks of rows and columns all together. Recently, (Wu, 2018) addressed this setting by developing an approximate matrix multiplication method based on block sampling. However, the method is inefficient, as it requires knowledge of optimal importance sampling probabilities that are expensive to compute.

We address this issue by showing that the method of Wu can be accelerated through the use of a randomized implicit trace estimation method. Doing so allows us to provably reduce the cost of sampling to near-linear in the size of the matrices being multiplied, without impacting the accuracy of the final approximate matrix multiplication. Overall, this yields a fast practical algorithm, which we test on a number of synthetic and real-world data sets. We complement our algorithmic contribution with the first extensive empirical comparison of block algorithms for randomized matrix multiplication. Our method offers a significant runtime advantage over the method of (Wu, 2018) and also outperforms basic uniform sampling of blocks. However, we find another recent method of (Charalambides, 2021) which uses sub-optimal but efficiently computable sampling probabilities often (but not always) offers the best trade-off between speed and accuracy.

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

**Keywords and phrases** Approximate matrix multiplication, randomized numerical linear algebra, trace estimation

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.103

**Supplementary Material** *Software (Source Code)*: `https://github.com/ChuhanYang/Hutch-AMM`

**Funding** This work was partially supported by National Science Foundation Award #2045590.

## 1 Introduction

Matrix computations are central across computing, with applications in optimization, scientific computing, data science, and more. In recent years, in an effort to tackle the challenge of scaling computations to larger and larger matrices, randomized methods have taken center stage [14, 27]. Collectively known as Randomized Numerical Linear Algebra or "RandNLA", the study of randomized matrix algorithms has led to faster algorithms for a number of central problems in linear algebra, including least squares regression [5, 24, 25], low-rank approximation [10, 13, 17, 23], trace estimation [18, 21], and more. Many of these algorithms are based on relatively simple sampling and sketching routines (like Johnson-Lindenstrauss random projection hashing-based methods) which are easily parallelized and adapted to modern computational environments, including distributed and streaming architectures [4].


## 1.1 Randomized Matrix Multiplication

Beyond the problems listed above, another topic of central interest in RandNLA is, of course, matrix multiplication. There has been significant work on developing randomized algorithms for matrix multiplication that return an approximate solution in a fraction of the time it would take to perform exact multiplication [6, 7]. Roughly, randomized matrix-multiplication methods can be split into two categories – random projection methods [25] and random sampling methods [12]. In both cases, the idea is to quickly compress given input matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$ to form matrix "sketches" $\mathbf{C} \in \mathbb{R}^{m\times c}$ and $\mathbf{D} \in \mathbb{R}^{c\times p}$, where $c \ll n$. We then return $\mathbf{CD}$ as an approximation to $\mathbf{AB}$. For many standard methods, including Johnson-Lindenstrauss random projection [25] and norm-based column and row sampling [12], a choice of $c = O(1/\epsilon^2)$ for $\epsilon \in (0, 1)$ ensures that with high probability:

$$\|\mathbf{AB} - \mathbf{CD}\|_F \leq \epsilon \|\mathbf{A}\|_F \|\mathbf{B}\|_F. \tag{1}$$

The approximation $\mathbf{CD}$ can be computed in $O(cmp) = O(mp/\epsilon^2)$ time, significantly improving on the naive cost of $O(nmp)$ when $n$ is large in comparison to $1/\epsilon^2$. So, we get an algorithmic speedup as long as $\mathbf{C}$ and $\mathbf{D}$ can be quickly obtained from $\mathbf{A}$ and $\mathbf{B}$.

### 1.1.1 The BasicMatrixMultiplication Method

In this paper, we are specifically interested in random sampling methods for computing $\mathbf{C}$ and $\mathbf{D}$. The most well-known method is the BasicMatrixMultiplication algorithm of Drineas, Kannan, and Mahoney [11, 12]. The idea behind the algorithm is elegant: the matrix multiplication $\mathbf{AB}$ can be cast as the sum of rank-one components, each an outer product of a column in $\mathbf{A}$ and the corresponding row in $\mathbf{B}$. In particular, $\mathbf{AB} = \sum_{i=1}^{n} \mathbf{A}^{(i)}\mathbf{B}_{(i)}$, where $\mathbf{A}^{(i)}$ denotes the $i^{\text{th}}$ column of $\mathbf{A}$ and $\mathbf{B}_{(i)}$ denotes the $i^{\text{th}}$ row of $\mathbf{B}$. BasicMatrixMultiplication approximates this sum by sampling the rank-one components using non-uniform sampling probabilities. To be more specific, we sample $\mathbf{A}^{(i)}\mathbf{B}_{(i)}$ with probability $p_i = \frac{\|\mathbf{A}^{(i)}\|_2\|\mathbf{B}_{(i)}\|_2}{\sum_{j=1}^{n}\|\mathbf{A}^{(j)}\|_2\|\mathbf{B}_{(j)}\|_2}$, so "heavy" columns and rows (which contribute more to the matrix product) are sampled with high probability. Equivalently, for the BasicMatrixMultiplication method, $\mathbf{C}$ and $\mathbf{D}$ consist of a subset of columns and rows from $\mathbf{A}$ and $\mathbf{B}$, sampled according to their $\ell_2$ norms. More details are provided in Section 2.

The BasicMatrixMultiplication algorithm is a powerful method with a number of desirable properties. First, the method achieves the strong bound of Equation (1), matching random projection methods in the worst case. However, on top of this:

1. When constructing $\mathbf{C}$ and $\mathbf{D}$, the BasicMatrixMultiplication method preserves sparsity and structure originally present in $\mathbf{A}$ and $\mathbf{B}$. This can lead to more compact sketches in practice. The same is not true of random projection based methods, which e.g. produce dense sketches $\mathbf{C}$ and $\mathbf{D}$, even if $\mathbf{A}$ and $\mathbf{B}$ are sparse.
2. The sampling probabilities $p_1, \ldots, p_n$, and thus the sketches $\mathbf{C}$ and $\mathbf{D}$ can be computed in just $O(mn + pn)$ time, which is linear in the size of the input. We simply need to compute the norm of all columns (resp., rows) in $\mathbf{A}$ (resp., $\mathbf{B}$).
3. The sampling probabilities used are *provably optimal* in the sense that they minimize the expected squared error $\mathbb{E}\left[\|\mathbf{AB} - \mathbf{CD}\|_F^2\right]$ amongst all choices of probabilities. As a result, BasicMatrixMultiplication often outperforms the worst-case bound of Equation (1).

Thanks to the advantages above, the BasicMatrixMultiplication algorithm has been widely adapted and applied to problems in information retrieval [15], image processing [20], and distributed computation [19].

## 1.2 Our Contributions

Despite its many virtues, an important practical issue with the BasicMatrixMultiplication method is that it samples columns and rows from $\mathbf{A}$ and $\mathbf{B}$ *independently at random.* In many applications, it would actually be preferable to sample contiguous *blocks* of columns from $\mathbf{A}$, and respective blocks of rows from $\mathbf{B}$, i.e., to sample every column $\mathbf{A}^{(i)}, \mathbf{A}^{(i+1)}, \ldots, \mathbf{A}^{(i+q-1)}$ for some starting index $i$ and block size $q$. The need for block sampling arises for a few reasons. First, in some settings it is desirable to keep nearby rows together to preserve structure in $\mathbf{A}$. As an example, [28] and [29] consider a problem where samples are obtained from block-structured finite element stiffness matrices. Moreover, even when $\mathbf{A}$ and $\mathbf{B}$ are unstructured, sampling multiple blocks of columns is typically more efficient on modern architectures where memory access costs are a major factor in determining final runtimes. When $\mathbf{A}$ and $\mathbf{B}$ are large enough that they must be stored in slow memory (e.g. on disk instead of in RAM) accessing a block of $q$ adjacent columns (which are stored adjacent on disk) will often be far cheaper than accessing a set of $q$ columns with non-adjacent indices.

### 1.2.1 The BlockBasicMatrixMultiplication Method

To address these concerns, recent work by Wu introduces a block-wise version of the BasicMatrixMultiplication method, which we call BlockBasicMatrixMultiplication [28]. The method maintains many of the nice properties of the BasicMatrixMultiplication method, and for any block size $q$, it can be shown that sampling $O(1/\epsilon^2)$ blocks results in the same error guarantee of Equation (1).[1] Moreover, it can be shown that the sampling probabilities used by Wu's method are *optimal for block sampling*, just as those used by BasicMatrixMultiplication were optimal for single column sampling. However, a major disadvantage of BlockBasicMatrixMultiplication is that these probabilities are no longer efficient to compute. Concretely, consider $\Theta_1, \ldots, \Theta_\ell$ which are disjoint subsets of adjacency indices that partition $\{1, \ldots, n\}$, i.e. for all $j$, $\Theta_i = \{k_i, k_i + 1, \ldots, k_i + q_i\}$ for some starting index $k_i$ and block size $q_i$, and $\Theta_1 \cup \ldots \cup \Theta_\ell = \{1, \ldots, n\}$. If we want to sample a subset of blocks from $\Theta_1, \ldots, \Theta_\ell$, [28] shows that the optimal probability to sample the $i^{\text{th}}$ block is equal to:

$$\hat{p}_i = \frac{\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F}{\sum_{j=1}^{\ell} \|\mathbf{A}^{(\Theta_j)}\mathbf{B}_{(\Theta_j)}\|_F} \tag{2}$$

This strictly generalizes the probabilities from BasicMatrixMultiplication, since for two vectors $\mathbf{A}^{(i)}$ and $\mathbf{B}_{(i)}$, $\|\mathbf{A}^{(i)}\mathbf{B}_{(i)}\|_F = \|\mathbf{A}^{(i)}\|_2 \|\mathbf{B}_{(i)}\|_2$. However, in the general block case, the probability is more expensive to compute. In particular, the naive cost of computing $\hat{p}_i$ equals $O(q_i mp)$ (the cost of multiplying $\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}$) so the total cost of sampling is $\sum_{i=1}^{\ell} O(q_i mp) = O(nmp)$. This is as slow as if we had computed $\mathbf{AB}$ exactly! Assuming for simplicity that $q_1 = \ldots = q_\ell = q$ (typically blocks have the same size), this runtime can be improved to $O(nmq + npq)$ when $q \leq p, m$ (see Equation (4) for more details). But nevertheless, the complexity is still greater than the linear run time of $O(nm + np)$ achieved by BasicMatrixMultiplication by a multiplicative factor of $q$. In practice, we might want $q$ in the 100s or 1000s to take sufficient advantage of memory locality.

[1] We might have hoped to obtain a *better* guarantee with block sampling. E.g. that when sampling blocks of size $q$, we would only need $O(\frac{1}{q\epsilon^2})$ block samples, which is a total of $O(1/\epsilon^2)$ columns. This is not possible, however, as can be seen by considering adversarial matrices with e.g. all but columns $1, q+1, 2q+1, \ldots$ set to zero. That said, in practice, sampling $c$ blocks from $\mathbf{A}$ and $\mathbf{B}$ often performs much better in terms of accuracy than sampling $c$ individual columns.

## 1.3 Faster Probabilities via Stochastic Trace Estimation

Our main contribution is to present a simple method for efficiently implementing the optimal sampling scheme of [28]. In particular, we show how to approximate the required sampling probabilities from Equation (2) up to a multiplicative constant factor using the classic Hutchinson's stochastic traces estimation method [18]. Another central technique in randomized numerical linear algebra, Hutchinson's traces estimator can approximate the $\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F$ term from Equation (2) with just a small number of matrix-vector multiplies with $\mathbf{A}^{(\Theta_i)}$ and $\mathbf{B}_{(\Theta_i)}$ involving randomly chosen vectors. The end result is a method running in $O((nm+np)\log(n))$ time for computing all probabilities, which is near linear in the size of the total size of the input matrices. At the same time, an analysis of our method shows that using approximate sampling probabilities yields the same theoretical guarantees as [28], up to constant factors. We call our method Hutchinson-estimated Block Approximate Matrix Multiplication (Hutch AMM for short).

### 1.3.1 Empirical Evaluation

We perform an extensive experimental evaluation of our Hutch AMM method, showing that, empirically, the probability computation can be performed to sufficient accuracy using just $5(nm + np)$ floating-point operations, even for very large values of $m$. The cost of then computing $\mathbf{CD}$ once the probabilities are computed and used for sampling is then a lower order computational. As a result, our method outperforms an efficient implementation of the approach from [28] in terms of runtime, without sacrificing accuracy in approximating $\mathbf{AB}$.

We also compare Hutch AMM to other baselines. For example, one natural approach is simply to uniformly sample from the rows and columns of $\mathbf{A}$ and $\mathbf{B}$ (either blocks or individual rows and columns). Uniform sampling typically requires more samples in comparison to using optimal importance sampling probabilities to obtain a given level of accuracy. The trade-off, however, is that uniform sampling has no computational overhead to compute probabilities – a sample of $c$ columns from $\mathbf{A}$ can be chosen in $O(c)$ time. The only major runtime cost is computing $\mathbf{CD}$. Nevertheless, we show that for block sampling, our method typically outperforms uniform sampling in terms of runtime to achieve a given level of approximation accuracy, showing the value of importance sampling.

Finally, we compare against a recent method of [3], which also studies block based sampling methods for approximate matrix multiplication. Their "Block CR Method" introduces an alternative approach that samples blocks with probabilities equal to:

$$\tilde{p}_i = \frac{\|\mathbf{A}^{(\Theta_i)}\|_F \|\mathbf{B}_{(\Theta_i)}\|_F}{\sum_{j=1}^{\ell} \|\mathbf{A}^{(\Theta_j)}\|_F \|\mathbf{B}_{(\Theta_j)}\|_F} \tag{3}$$

While not optimal for minimizing $\mathbb{E}\left[\|\mathbf{AB} - \mathbf{CD}\|_F^2\right]$, these probabilities minimize a natural upper bound[2] on the expected squared error, and can be shown to achieve the bound of Equation (1) with $q = O(1/\epsilon^2)$ samples [3]. So, they match the result of Wu in the worst case. At the same time, $\tilde{p}_1, \ldots, \tilde{p}_\ell$ can be computed in linear time, $O(nm + np)$, as they do not require multiplying the blocks $\mathbf{A}^{(\Theta_i)}$ and $\mathbf{B}_{(\Theta_i)}$.

[2] The text of [3] implies that the probabilities are optimal, but there is a small error in the derivation of the variance of the estimator considered, where an upper bound is mistakenly considered to be an equality. Nevertheless, the authors conclusion, that $O(1/\epsilon^2)$ block samples suffice to achieve error $\|\mathbf{AB} - \mathbf{CD}\|_F \leq \epsilon\|\mathbf{A}\|_F\|\mathbf{B}\|_F$, still holds.

Overall, we find the Block CR method of [3] extremely effective. While it does not yield as good an accuracy for a given number of samples as our method and that of Wu, the difference is relatively small. So, when comparing overall runtime to approximate the matrix product $\mathbf{AB}$, the method of [3] usually offers the best accuracy vs. runtime trade-off.

### 1.4 Paper Organization

The organization of this paper is as follows: In Section 2, we introduce the BlockBasicMatrixMultiplication and our proposed modification. Section 3 presents a detailed analysis and comparison of the expected squared error of the methods. In Section 4, we provide the experiment details, followed by a brief discussion of the results. Finally, we conclude the paper and discuss future research directions in Section 5.

## 2 Methodology

### 2.1 Notation

We denote matrices and vectors using bold Roman letters. For a vector $\mathbf{v} \in \mathbb{R}^n$, $\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^n \mathbf{v}_i}$ denotes the standard Euclidean norm. For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n \mathbf{A}_{ij}}$ denotes the Frobenius norm. We use superscript $\mathbf{A}^{(i)}$ to denote the $i^{\text{th}}$ column of $\mathbf{A}$ and subscript $\mathbf{A}_{(j)}$ to denote the $j^{\text{th}}$ row. For a set of $c$ indices $S$, we let $\mathbf{A}^{(S)}$ denote the $m \times c$ matrix containing $\mathbf{A}^{(j)}$ for $j \in \mathcal{S}$. Similarly, $\mathbf{A}_{(S)}$ denote the $c \times n$ matrix containing $\mathbf{A}_{(j)}$ for $j \in \mathcal{S}$. For a square $\mathbf{M} \in \mathbb{R}^{n\times n}$, $\text{tr}(\mathbf{M}) = \sum_{i=1}^n \mathbf{M}_{ii}$ denotes the trace. We use $\Pr[B]$ to denote the probability of a random event $B$ and $\mathbb{E}[X]$ to denote the expectation of a random variable $X$.

### 2.2 Hutchinson-estimated block Approximate Matrix Multiplication

Given matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$, BasicMatrixMultiplication returns an unbiased estimator for the matrix product $\mathbf{AB}$ by sampling rank-one components (each is an outer product of a column from $\mathbf{A}$ and corresponding row from $\mathbf{B}$). Specifically, the algorithm selects and re-weights a subset of $c$ columns from $\mathbf{A}$ to form a matrix $\mathbf{C} \in \mathbb{R}^{m\times c}$ and the corresponding $c$ rows from $\mathbf{B}$ to form a matrix $\mathbf{D} \in \mathbb{R}^{c\times p}$ so that $\mathbf{CD} \approx \mathbf{AB}$. To reduce the approximation error, rows and columns are sampled with non-uniform probabilities, and appropriately scaled after sampling by the inverse probability to ensure the $\mathbb{E}[\mathbf{CD}] = \mathbf{AB}$. The algorithm is summarized in Algorithm 1:

**Algorithm 1** BasicMatrixMultiplication w/ Optimal Sampling Probability [12].

**Input:** $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{n\times p}$, number of samples $c$.
**Output:** Estimate of matrix product $\mathbf{AB}$.

For all $i \in 1, \ldots, n$, compute optimal sampling probability $p_i = \frac{\|\mathbf{A}^{(i)}\|_2 \|\mathbf{B}_{(i)}\|_2}{\sum_{j=1}^n \|\mathbf{A}^{(j)}\|_2 \|\mathbf{B}_{(j)}\|_2}$.
**for** $t = 1 \ldots, c$ **do**
  Pick $j \in \{1, \ldots, n\}$ with probability $\Pr(j = k) = p_k$.
  Set $\mathbf{C}^{(t)} = \frac{\mathbf{A}^{(j)}}{\sqrt{cp_j}}$ and $\mathbf{D}_{(t)} = \frac{\mathbf{B}_{(j)}}{\sqrt{cp_j}}$
**end**
Return $\mathbf{CD}$

As discussed in Section 1, the sampling probabilities $p_1, \ldots, p_n$ used in Algorithm 1 are optimal in that they minimize $\mathbb{E}\left[\|\mathbf{AB} - \mathbf{CD}\|_F^2\right]$ amongst all possible choices of probabilities [12] (when using an unbiased estimator of the same form as Algorithm 1).

It is possible to extend the BasicMatrixMultiplication method to sampling blocks of rows and columns. To do so, we consider a partition of the indices $\{1, \ldots, n\}$ into $\ell$ disjoint sets $\Theta_1, \ldots, \Theta_\ell$. [28] derived optimal probabilities $\hat{p}_1, \ldots, \hat{p}_\ell$ for sampling in the block setting, which are shown below in Algorithm 2. Again, these probabilities minimize $\mathbb{E}\left[\|\mathbf{AB} - \mathbf{CD}\|_F^2\right]$ amongst all possible choices of sampling probabilities.

**Algorithm 2** Block BasicMatrixMultiplication w/ Optimal Sampling Probability [28].

**Input:** $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{n\times p}$, number of samples $c$, disjoint set of column indices $\{\Theta_1, \ldots, \Theta_\ell\}$ with $\Theta_1 \cup \ldots \cup \Theta_\ell = \{1, \ldots, n\}$.

**Output:** Estimate of matrix product $\mathbf{AB}$.

For all $i \in 1, \ldots, \ell$, compute optimal sampling probability $\hat{p}_i = \frac{\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F}{\sum_{j=1}^{\ell} \|\mathbf{A}^{(\Theta_j)}.\mathbf{B}_{(\Theta_j)}\|_F}$

Initialize $\mathbf{C}$ and $\mathbf{D}$ as empty matrices.

**for** $t = 1, \ldots, c$ **do**

  Pick $j \in \{1, \ldots, \ell\}$ with probability $\Pr(j = k) = \hat{p}_k$.

  Append $\frac{\mathbf{A}^{(\Theta_j)}}{\sqrt{c\hat{p}_j}}$ onto $\mathbf{C}$'s columns and $\frac{\mathbf{B}_{(\Theta j)}}{\sqrt{c\hat{p}_j}}$ onto $\mathbf{D}$'s rows.

**end**

Return $\mathbf{CD}$

▷ Claim 1. Assume $\Theta_1, \ldots, \Theta_\ell$ are equally sized, so each contains $q = \frac{n}{l}$ indices. Then the BlockBasicMatrixMultiplication method of Algorithm 2 can be implemented in time $O(mnp)$, or in time $O(n(m+p)q + cqmp)$, which is faster when $q \leq m, p$.

Proof. Once sampling probabilities are computed and sampling performed, the cost of multiplying $\mathbf{CD}$ is equal to $O(cqmp)$. Specifically, $\mathbf{C}$ contains $c$ blocks of $q$ columns, so has dimension $m \times cq$ and $\mathbf{D}$ has dimension $cq \times p$. We focus on the cost of computing the optimal probabilities, $\hat{p}_1, \ldots, \hat{p}_\ell$. Doing so requires computing $\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F$ all $i$. Since $\mathbf{A}^{(\Theta_i)} \in \mathbb{R}^{m\times q}$ and $\mathbf{B}_{(\Theta_i)} \in \mathbb{R}^{q\times p}$, this can be done directly in $O(qmp)$ time. Summing over all $\ell$ blocks and using that $q = \frac{n}{\ell}$, the total runtime is $O(nmp)$ to compute $p_1, \ldots, p_\ell$. Alternatively, if $q \leq m + p$, we can use the cyclic property of the trace:

$$\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F^2 = \mathrm{tr}(\mathbf{B}_{(\Theta_i)}^T(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}) = \mathrm{tr}((\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\mathbf{B}_{(\Theta_i)}^T). \quad (4)$$

If we first compute $(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}$ and $\mathbf{B}_{(\Theta_i)}\mathbf{B}_{(\Theta_i)}^T$ and then multiply the resulting $q \times q$ matrices, the above trace can be computed in time $O(q^2m + q^2p + q^3) = O((m+p)q^2)$. Again, summing over all $\ell$ blocks and using that $q = \frac{n}{\ell}$, the total runtime is $O(n(m+p)q)$. ◁

As illustrated by Claim 1, the computational cost of BlockBasicMatrixMultiplication can be prohibitively expensive. When $q$ is larger than either $m$ or $p$, the cost is as large as the cost of computing the product $\mathbf{AB}$ exactly.

Our proposed approach is to speed up the BlockBasicMatrixMultiplication method by using Hutchinson's stochastic trace estimator [16, 18] to efficiently approximate $\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F^2 = \mathrm{tr}(\mathbf{B}_{(\Theta_i)}^T(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)})$. Given an $p \times p$ matrix $\mathbf{M}$, this estimator estimates $\mathrm{tr}(\mathbf{M})$ via repeated multiplication of the matrix by random vectors. In particular, for a sampling parameter $h$ and vectors $\mathbf{g}_1, \ldots, \mathbf{g}_h$ each chosen to have independent random entries with mean 0 and variance 1, the estimator takes the form:

$$H_h(\mathbf{M}) = \frac{1}{h}\sum_{j=1}^{h} g_j^\intercal \mathbf{M} g_j. \tag{5}$$

Typically, random $\{-1, +1\}$ Rademacher random variables are chosen for the entries of each $\mathbf{g}_j$. It is easily checked that $\mathbb{E}\left[H_h(\mathbf{M})\right] = \mathrm{tr}(\mathbf{M})$ and for sufficiently large $h$, the estimator concentrates around its expectation [1, 8]. The value of the estimation in Equation (5) is that, for $\mathbf{M} = \mathbf{B}_{(\Theta_i)}^T(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}$, each term in the sum $\sum_{i=1}^{h} g_j^\intercal \mathbf{M} g_j$ can be computed in just $O(pq + mq + pq + mq) = O(q(m+p))$ time by multiplying $\mathbf{B}_{(\Theta_i)}^T(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\mathbf{g}_j$ from right to left. When $h$ is a small constant, the cost is thus linear in the total size of the blocks $\mathbf{A}^{(\Theta_i)}$ and $\mathbf{B}_{(\Theta_i)}$. Overall, our proposed algorithm is summarized in Algorithm 3:

**Algorithm 3** Hutchinson-estimated Block Approx. Matrix Multiplication (Hutch AMM).

**Input:** $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{B} \in \mathbb{R}^{n\times p}$, number of samples $c$, disjoint sets of column indices $\{\Theta_1, \ldots, \Theta_\ell\}$ with $\Theta_1 \cup \ldots \cup \Theta_\ell = \{1, \ldots, n\}$, number of Hutchinson's samples $h$.

**Output:** Estimate of matrix product $\mathbf{AB}$.

For all $i \in 1, \ldots, \ell$, compute approximate optimal sampling probability

$$\bar{p}_i = \frac{\sqrt{H_h(\mathbf{B}_{(\Theta_i)}^T(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)})}}{\sum_{j=1}^{\ell}\sqrt{H_h(\mathbf{B}_{(\Theta_j)}^T(\mathbf{A}^{(\Theta_j)})^T\mathbf{A}^{(\Theta_j)}\mathbf{B}_{(\Theta_j)})}}$$

Initialize $\mathbf{C}$ and $\mathbf{D}$ as empty matrices.

**for** $t = 1, \ldots, c$ **do**

  Pick $j \in \{1, \ldots, \ell\}$ with probability $\Pr(j = k) = \bar{p}_k$.

  Append $\frac{\mathbf{A}^{(\Theta_j)}}{\sqrt{c\bar{p}_j}}$ onto $\mathbf{C}$'s columns and $\frac{\mathbf{B}_{(\Theta_j)}}{\sqrt{c\bar{p}_j}}$ onto $\mathbf{D}$'s rows.

**end**

Return $\mathbf{CD}$

We can bound the runtime of Algorithm 3 as follows:

▷ Claim 2. Assume $\Theta_1, \ldots, \Theta_\ell$ are equally sized, so each contains $q = \frac{n}{l}$ indices. Then the Hutch AMM method of Algorithm 3 can be implemented in time $O(nh(m+p) + cqmp)$.

Proof. The proof follows from the discussion above. For each block, computing the Hutchinson's estimate $H_h(\mathbf{B}_{(\Theta_i)}^T(\mathbf{A}^{(\Theta_i)})^T\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)})$ takes time $O(qh(m+p))$ for a total runtime of $\ell \cdot O(qh(m+p)) = O(nh(m+p))$. We then have an additional cost of $O(cqmp)$ to compute $\mathbf{CD}$, which will typically be a lower order term when the number of samples $c$ is small. ◁

From Claim 1 and Claim 2, we can check that Hutch AMM improves on the BlockBasicMatrixMultiplication algorithm whenever the number of Hutchinson's iterations $h$ is less than the block size $q$. As we will prove in the next section, if suffices to set $h = O(\log n)$ to match the accuracy guarantees of BlockBasicMatrixMultiplication up to a multiplicative constant.

## 3 Error Analysis

In this section we provide an analysis of our Hutch AMM method. We start by stating the main result from [28], which bounds the expected squared error of the BlockBasicMatrixMultiplication method.

▶ **Proposition 3** (Proposition 2.2 [28]). *For input matrices* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{B} \in \mathbb{R}^{n\times p}$*, let* $\mathbf{CD}$ *be the approximate matrix product returned by Algorithm 2. We have that* $\mathbb{E}[\mathbf{CD}] = \mathbf{AB}$ *and the expected squared approximation error in Frobenius norm is:*

$$\mathbb{E}[\|\mathbf{AB} - \mathbf{CD}\|_F^2] = \frac{1}{c}\sum_{i=1}^{\ell}\frac{1}{\hat{p}_i}\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F^2 - \frac{\|\mathbf{AB}\|_F^2}{c}. \tag{6}$$

*Setting* $\hat{p}_i := \frac{\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F}{\sum_{i=1}^{l}\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F}$ *for* $i = 1, \ldots, \ell$*, we obtain minimum expected error, which can be upper bounded by:*

$$\mathbb{E}[\|\mathbf{AB} - \mathbf{CD}\|_F^2] \leq \frac{1}{c}\left(\sum_{i=1}^{\ell}\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F\right)^2 \tag{7}$$

Note that, applying submultiplicativity of the Frobenius norm and Cauchy-Schwarz inequality we can (loosely) bound:

$$\begin{aligned}\frac{1}{c}\left(\sum_{i=1}^{\ell}\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F\right)^2 &\leq \frac{1}{c}\left(\sum_{i=1}^{\ell}\|\mathbf{A}^{(\Theta_i)}\|_F\|\mathbf{B}_{(\Theta_i)}\|_F\right)^2 \\ &\leq \frac{1}{c}\left(\sum_{i=1}^{\ell}\|\mathbf{A}^{(\Theta_i)}\|_F^2\right)\left(\sum_{i=1}^{\ell}\|\mathbf{B}^{(\Theta_i)}\|_F^2\right) = \frac{1}{c}\|\mathbf{A}\|_F^2\|\mathbf{B}\|_F^2\end{aligned}$$

Using the resulting bound on $\mathbb{E}[\|\mathbf{AB} - \mathbf{CD}\|_F^2]$ we can apply Markov's inequality to the non-negative random variable $\|\mathbf{AB} - \mathbf{CD}\|_F^2$ to prove that, with probability $(1 - \delta)$, BlockBasicMatrixMultiplication achieves error $\|\mathbf{AB} - \mathbf{CD}\|_F \leq \frac{1}{\sqrt{\delta c}}\|\mathbf{A}\|_F\|\mathbf{B}\|_F$. In other words, with $c = O(1/\delta\epsilon^2)$ samples, we obtain error $\epsilon\|\mathbf{A}\|_F\|\mathbf{B}\|_F$ with probability $(1 - \delta)$, as desired.

What happens if we sample with approximately optimal probabilities? From Equation (6), it can be checked directly that, if instead $\mathbf{C}$ and $\mathbf{D}$ are sampled with probabilities $\bar{p}_1, \ldots, \bar{p}_\ell$ satisfying $\bar{p}_i \geq \frac{1}{\beta}\hat{p}_i$ for some constant $\beta \geq 1$, then the expected squared error can be bounded by:

$$\mathbb{E}[\|\mathbf{AB} - \mathbf{CD}\|_F^2] \leq \frac{\beta}{c}\left(\sum_{i=1}^{\ell}\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F\right)^2. \tag{8}$$

In other words, as long as $\bar{p}_1, \ldots, \bar{p}_\ell$ approximate the optimal sampling probabilities $\hat{p}_1, \ldots, \hat{p}_\ell$ up to a constant factor, we obtain expected squared error that is within a multiplicative constant of the bound from Equation (8) achieved by Wu's optimal method.

With this in mind, we focus on showing that the approximate probabilities used in our Hutch AMM method satisfy this multiplicative bound. In particular, we prove:

▶ **Lemma 4.** *For inputs* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and* $\mathbf{B} \in \mathbb{R}^{n\times p}$*, let* $\bar{p}_1, \ldots, \bar{p}_\ell$ *be computed as in Algorithm 3 with parameter* $h$*, and let* $\hat{p}_1, \ldots, \hat{p}_\ell$ *be the optimal sampling probabilities from Algorithm 2. As long as* $h \geq C\log(\ell/\delta)$ *for a fixed constant* $C$*, then with probability* $1 - \delta$*,*

$$\bar{p}_i \geq \frac{1}{2}\hat{p}_i \qquad \text{for all } i \in 1, \ldots \ell.$$

In other words, as long as $h = O(\log \ell) \leq O(\log n)$, then with high probability we obtain sampling probabilities with a constant factor of the optimal probabilities, and thus expected squared error with a constant of Equation (7), as desired.

To prove 4, we use the following standard bound on the accuracy of Hutchinson's estimator:

▶ **Lemma 5** (See [8] or [21]). *Let $\mathbf{M} \in \mathbb{R}^{d\times d}$ and $\delta \in (0, 1/2]$. Let $H_h(\mathbf{M})$ be Hutchinson's estimator run for $h$ repetition, as in Equation (5). For fixed constants $C_1, C_2$, if $h > C_1 \log(1/\delta)$, then with probability $\geq 1-\delta$*

$$|H_h(\mathbf{M}) - \mathrm{tr}(\mathbf{M})| \leq C_2\sqrt{\frac{\log(1/\delta)}{h}}\|\mathbf{M}\|_F. \tag{9}$$

**Proof of Lemma 4.** Let $\mathbf{X}_i$ denote $\mathbf{X}_i = \mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}$. Let $h = 4\max(C_1, C_2^2)\log(\ell/\delta)$. Then by Lemma 5 it follows that, for any $i \in 1, \ldots, \ell$,

$$\left|H_h(\mathbf{X}_i^T\mathbf{X}_i) - \mathrm{tr}(\mathbf{X}_i^T\mathbf{X}_i)\right| \leq \frac{1}{2}\|\mathbf{X}_i^T\mathbf{X}_i\|_F,$$

with probability $1 - \delta/\ell$. By a union bound, the statement holds simultaneously for all $i \in 1, \ldots, \ell$ with probability $1-\delta$. Since $\mathbf{X}_i^T\mathbf{X}_i$ is a positive semidefinite matrix, we have that $\|\mathbf{X}_i^T\mathbf{X}_i\|_F \leq \mathrm{tr}(\mathbf{X}_i^T\mathbf{X}_i)$. So we can conclude that for all $i \in 1, \ldots, \ell$,

$$.5 \cdot \mathrm{tr}(\mathbf{X}_i^T\mathbf{X}_i) \leq H_h(\mathbf{X}_i^T\mathbf{X}_i) \leq 1.5 \cdot \mathrm{tr}(\mathbf{X}_i^T\mathbf{X}_i).$$

We have that $\bar{p}_i = \frac{\sqrt{H_h(\mathbf{X}_i^T\mathbf{X}_i)}}{\sum_{j=1}^{\ell}\sqrt{H_h(\mathbf{X}_j^T\mathbf{X}_j)}}$, so we conclude that, as desired,

$$\bar{p}_i \geq \frac{\sqrt{.5}\cdot\mathrm{tr}(\mathbf{X}_i^T\mathbf{X}_i)}{\sqrt{1.5}\sum_{j=1}^{\ell}\mathrm{tr}(\mathbf{X}_j^T\mathbf{X}_j)} \geq \frac{1}{2}\hat{p}_i. \qquad \blacktriangleleft$$

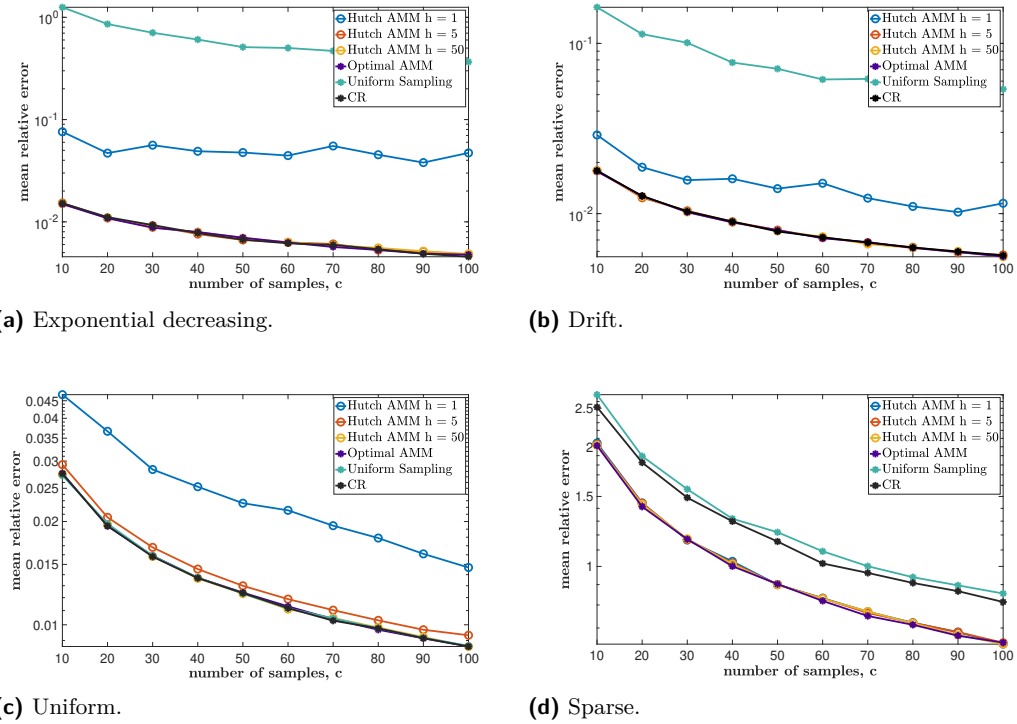
Combined with the bound in Equation (8) and Markov's inequality, Lemma 4 implies that as long as $h = O(\log(\ell/\delta)$, then with probability $1-\delta$, Hutch AMM returns an estimate $\mathbf{CD}$ satisfying $\|\mathbf{AB} - \mathbf{CD}\|_F \leq \frac{1}{\sqrt{\delta c}}\sum_{i=1}^{\ell}\|\mathbf{A}^{(\Theta_i)}\mathbf{B}_{(\Theta_i)}\|_F \leq \frac{1}{\sqrt{\delta c}}\|\mathbf{A}\|_F\|\mathbf{B}\|_F$. Accordingly, we match the guarantees of the BlockBasicMatrixMultiplication method, but with a total computational cost of just $O(n\log(n)(m+p))$ to compute sampling probabilities, which is near linear in the size of the inputs $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times p}$.

## 4 Experiments

In this section, we test our algorithm on both synthetic matrices and real-world datasets. We demonstrate its performance by comparing it with the optimal BlockBasicMatrixMultiplication method [28], which in this section we call "Optimal Block AMM" for conciseness. As discussed in the introduction, we also compare with two baseline method: sampling blocks using uniform probabilities $\tilde{p}_i = \frac{1}{\ell}$ and the CR method from [3], which samples with the probabilities shown in Equation (3).[3]

To compare methods, we use mean relative error $\frac{\|\mathbf{CD}-\mathbf{AB}\|_F}{\|\mathbf{AB}\|_F}$ and the total computational time as two performance metrics in estimation accuracy and efficiency. All experiments were conducted in Matlab R2022b on a laptop with a 2.7 GHz Quad-Core Intel Core i7 and 16 GB memory.

[3] We note that there has been some recent follow-up work in [22] on modifications of the BlockBasicMatrixMultiplication method. However, the algorithms in that work ultimately perform individual row/column sampling within each block, so they are not directly comparable to the other methods studied in this paper, which always sampled contiguous blocks of rows or columns together.




**(a)** Exponential decreasing.

**(b)** Drift.

**(c)** Uniform.

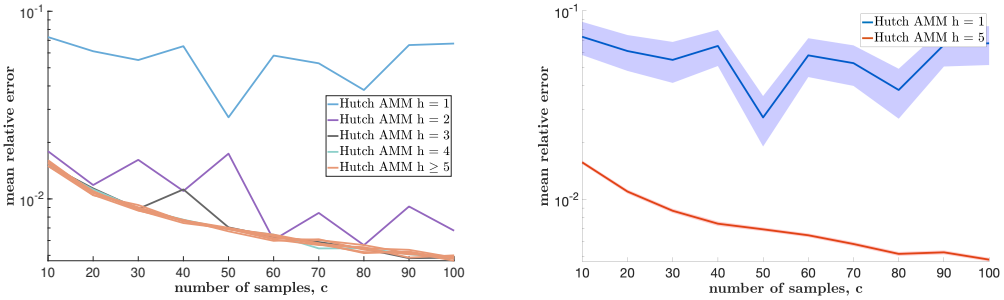
**(d)** Sparse.

**Figure 1** Performance comparison of different block approximate matrix multiplication methods on matrices generated by different procedures. The matrices for plots (a-c) have size of $100 \times 10000$, with a sampling block size $q = 100$, (d) involves matrices of size of $10 \times 100000$, with sampling block size $q = 1000$. Our results show that, when comparing the number of samples $c$ to average relative error for 200 trials, the Hutch AMM method with $h \geq 5$ consistently performs as well as the optimal Optimal Block AMM method for matrices with significant differences in row/column norms (a,b), while the Hutch AMM with $h = 1$ and uniform sampling methods perform worse. The CR methods perform similarly to the optimal block AMM, except for the sparse case (d).

## 4.1 Data sets

For our synthetic data, we generate $\mathbf{B} \in \mathbb{R}^{10000\times 100}$ with all entries uniform random in $[0, 1]$. For two data sets, we generate $\mathbf{A} \in R^{100\times 10000}$ with random Gaussian entries with variance 1, and mean depending on the column index, $i$. For (a) Exponential decreasing, the means are uniformly spaced on an exponential grid from $\exp(50)$ down to 1. For (b) Drift, the means follow two trends. For the first 5000 columns they are exponentially decreasing as in (a), and for the next 5000 columns the means are linearly decreasing. A similar matrix was used in [9]. These first two data sets have columns and blocks of columns with widely varying magnitudes. For our third dataset, (c) Uniform, $\mathbf{A}$ is simply generated with entries uniform random in $[0, 1]$ like $\mathbf{B}$. Finally, for (d) Sparse, we construct a sparse matrix $\mathbf{A} \in \mathbb{R}^{10\times 100000}$ with 0.1% non-zero entries selecting uniformly at random in $[0, 1]$. We construct $\mathbf{B} \in \mathbb{R}^{100000\times 10}$ with entries drawn from a standard normal distribution, but with 1% entries randomly replaced with larger values uniformly distributed between 0 and an arbitrary positive number (we used $\exp(4)$ in this paper).

For real-world data, we consider two different application scenarios: (1) Natural language processing: We extracted TF-IDF matrices from a subset of the TDT2 corpus (Nist Topic Detection and Tracking corpus)[2]. (2) Time series: We formed the whole-month (February




**(a)** Mean relative error comparison of Hutch AMM from h=1 to h=10.

**(b)** Estimation error for selected h plotted in shaded region, shaded region ranges between plus and minus one standard deviation from mean.

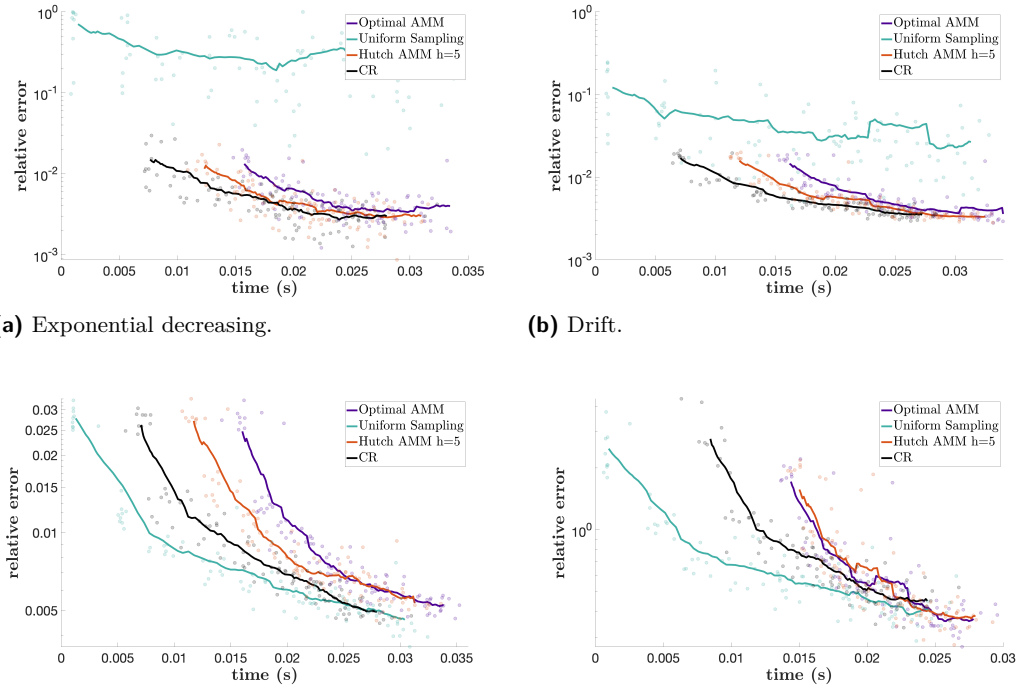
**Figure 2** This figure investigates the impact of $h$ on the estimation accuracy of Hutch AMM methods. The graph shows the average relative estimation error against sampling number $c$ for 200 trials for the Exponential Decreasing synthetic data. Higher choices of $h$ generally lead to better average error, but there is little to no benefit of choosing a value of $h$ larger than 5. Similarly, low choices of $h$ lead to higher variance, as shown in (b). The shaded region represents the plus and minus one standard deviation from the mean relative estimation error among the 200 trials. As we can see, this region is much narrower for $h = 5$ in comparison to $h = 1$.

2022) yellow taxi trip records as a matrix using NYC Taxi and Limousine Commission (TLC) Trip Record Data[26], the columns are sorted by pick-up time. We include numerical features such as trip distances, itemized fares, and driver-reported passenger counts, as well as one-hot-encoded categorical features such as VendorID and payment type. For the TF-IDF experiments, we construct a matrix $\mathbf{A}$ of size $1000 \times 36763$; for the trip record experiments, $\mathbf{A}$ has size $24 \times 2877693$. In both cases we set $\mathbf{B} = \mathbf{A}^T$ and use block size $q = 100$.

## 4.2 Results

We first compare the accuracy of block approximate matrix multiplication methods as a function of the number of samples $c$. As shown in Figure 1, our Hutch AMM method run with $h$ as small as 5 consistently matches the Optimal Block AMM method from [28]. Both methods outperform uniform sampling, except when $\mathbf{A}$ and $\mathbf{B}$ have uniform column norms, in which case there is limited value in importance sampling. The CR method also performs quite well, although is worse than our Hutch AMM method for the sparse synthetic data. Running Hutch AMM with $h = 1$ shows worse performance. In Figure 2, we take a closer look at the impact of varying $h$, and the choice of $h = 5$ seems to be a sweet spot – lower values lead to higher error (due to worse approximation of the optimal sampling probabilities) whereas high values offer little improvement, and lead to higher computation cost.

Having settled on $h = 5$ as a default parameter for our Hutch AMM method, we proceed to compare the runtime against all baselines. We do so in Figure 3 for the synthetic data and Figure 4 for the real-world data sets. In both cases, we perform repeated trials with various choices of $c$. Since runtime and accuracy are not deterministic functions of $c$, this leads to a scatter plot comparing running time vs. accuracy, which we visualize plot plotting a moving average trend for each method tested. Overall, the plots show that Hutch AMM offers improved computation time over Optimal Block AMM, and typically much better accuracy than uniform sampling. However, while it showed better performance than the CR method in terms of accuracy for a given number of samples for some problems (e.g. the Sparse data), the slightly lower computational complexity of the CR method comes through




**(a)** Exponential decreasing.

**(b)** Drift.

**(c)** Uniform.

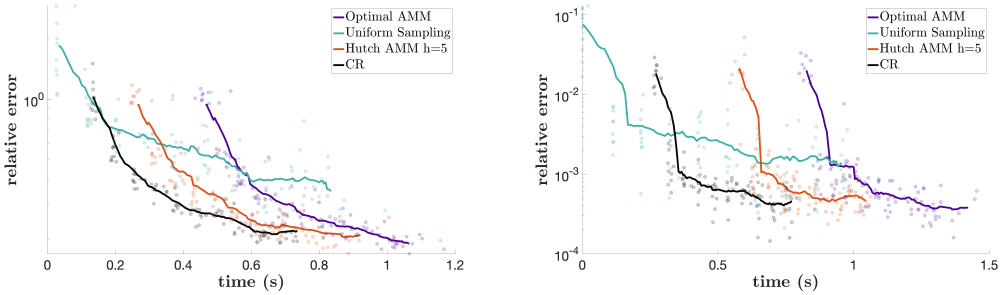
**(d)** Sparse.

**Figure 3** Relative estimation error against computation time, with a moving average line generated using window length 20. Matrices are generated using the same procedures as in Figure 1. For interpretation: A line to the left indicates the corresponding method's superior computational efficiency; a line towards the bottom indicates the corresponding method's superior estimation accuracy. Our observations indicate that both our Hutch AMM method and the CR method improve on the Optimal Block Matrix multiplication method in terms of efficiency/accuracy tradeoff. They also beat uniform sampling in all cases except for (c). Overall, the CR method shows the best overall performance, despite its use of non-uniform sampling probabilities.

in the plots. In terms of overall computational complexity, in our experiments, it was more efficient to run the CR method. The lack of optimal sampling probabilities is made up for by including a slightly larger number of samples $c$ in $\mathbf{C}$ in $\mathbf{D}$.

## 5 Summary and Conclusion

In this study, we present the Hutchinson-estimated block Approximate Matrix Multiplication (Hutch AMM) method, which is a more computationally efficient variant of the existing BlockBasicMatrixMultiplication method. Our proposed method utilizes Hutchinson's estimator to calculate a near-optimal sampling probability, which reduces computation cost while maintaining a high level of estimation accuracy when an appropriate $h$ parameter is chosen. We validate our method through a detailed complexity analysis and proof of bounded squared error. Additionally, we perform experiments on both synthetic and practical data. Our results demonstrate that the Hutch AMM method is more computationally efficient and provides accurate estimation on par with the optimal BlockBasicMatrixMultiplication in all cases. Furthermore, we show that our proposed method can offer better performance in terms of estimation accuracy than the similarly efficient CR block matrix multiplication method.




**(a)** TDT2 corpus. **(b)** NYC Yellow Taxi Trip.

**Figure 4** The figure shows scatter points of relative estimation error plotted against computation time, with a moving average line generated using smoothing window length 20. The matrices used are practical datasets: (a) TF-IDF matrices from a subset of the TDT2 corpus, and (b) the February 2022 Trip Record Data from the NYC Taxi and Limousine Commission (TLC). Despite the different characteristics of these datasets, we still observe that Hutch AMM is more computationally efficient and achieves close estimation accuracy to optimal block AMM.

We acknowledge that our proposed Hutch AMM method has potential for improvement. Although our current partitioning strategy splits matrix columns into blocks by a natural sequence, it may be possible to optimize this process by implementing alternative partitioning strategies tailored for specific matrix factors. Moreover, we can explore alternative estimators for near-optimal sampling probabilities or even consider biased estimators to further enhance the computational efficiency of AMM. These are promising avenues for future research in this area.

## References


1 Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2), 2011.

2 Deng Cai, Xuanhui Wang, and Xiaofei He. Probabilistic dyadic data analysis with local and global consistency. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*, pages 105–112, 2009.

3 Neophytos Charalambides, Mert Pilanci, and Alfred O Hero. Approximate weighted CR coded matrix multiplication. In *Proceedings of the 2021 International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 5095–5099. IEEE, 2021.

4 Kenneth Clarkson and David Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 205–214, 2009.

5 Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90, 2013.

6 Edith Cohen and David D Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30(2):211–252, 1999.

7 Michael B. Cohen, Jelani Nelson, and David P. Woodruff. Optimal approximate matrix product in terms of stable rank. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP)*, 2016.

8 Alice Cortinovis and Daniel Kressner. On randomized trace estimates for indefinite matrices with an application to determinants. *Foundations of Computational Mathematics*, 22(3):875–903, 2022.

**9** Amey Desai, Mina Ghashami, and Jeff M Phillips. Improved practical matrix sketching with guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 28(7):1678–1690, 2016.

**10** Amit Deshpande and Santosh Vempala. Adaptive sampling and fast low-rank matrix approximation. In *Proceedings of the 10th International Workshop on Randomization and Computation (RANDOM)*, pages 292–303, 2006.

**11** Petros Drineas and Ravi Kannan. Fast Monte-Carlo algorithms for approximate matrix multiplication. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 452–459. IEEE, 2001.

**12** Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006.

**13** Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

**14** Petros Drineas and Michael W. Mahoney. RandNLA: Randomized numerical linear algebra. *Commun. ACM*, 59(6), 2016.

**15** Sylvester Eriksson-Bique, Mary Solbrig, Michael Stefanelli, Sarah Warkentin, Ralph Abbey, and Ilse CF Ipsen. Importance sampling for a Monte Carlo matrix multiplication algorithm, with application to information retrieval. *SIAM Journal on Scientific Computing*, 33(4):1689–1706, 2011.

**16** Didier Girard. Un algorithme simple et rapide pour la validation croisee géénéralisée sur des problémes de grande taille. Technical report, École nationale supérieure d'informatique et de mathématiques appliquées de Grenoble, 1987.

**17** Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

**18** Michael F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

**19** Tayyebeh Jahani-Nezhad and Mohammad Ali Maddah-Ali. CodedSketch: A coding scheme for distributed computation of approximated matrix multiplication. *IEEE Transactions on Information Theory*, 67(6):4185–4196, 2021.

**20** Humberto Madrid, Valia Guerra, and Marielba Rojas. Sampling techniques for Monte Carlo matrix multiplication with applications to image processing. In *Mexican Conference on Pattern Recognition*, pages 45–54. Springer, 2012.

**21** Raphael A Meyer, Cameron Musco, Christopher Musco, and David P Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 142–155. SIAM, 2021.

**22** Chengmei Niu and Hanyu Li. Optimal sampling algorithms for block matrix multiplication. *Journal of Computational and Applied Mathematics*, page 115063, 2023.

**23** Christos H Papadimitriou, Prabhakar Raghavan, Hisao Tamaki, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235, 2000.

**24** Vladimir Rokhlin and Mark Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, 2008.

**25** Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 143–152, 2006.

**26** NYC Taxi and Limousine Commission (TLC). Tlc trip record data. `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`, 2022.

27 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014.

28 Yue Wu. A note on random sampling for matrix multiplication. *arXiv preprint*, 2018. arXiv:1811.11237.

29 Yue Wu, Dimitris Kamilis, and Nick Polydorides. A randomised finite element method for elliptic partial differential equations. *arXiv preprint*, 2019. arXiv:1903.07696.

# A Simple Boosting Framework for Transshipment

**Goran Zuzic**
Google Research, Zürich, Switzerland
ETH Zürich, Switzerland

## Abstract

Transshipment is an important generalization of both the shortest path problem and the optimal transport problem. The task asks to route a demand using a flow of minimum cost over (uncapacitated) edges. Transshipment has recently received extensive attention in theoretical computer science as it is the centerpiece of all modern theoretical breakthroughs in parallel and distributed (approximate) shortest-path computation, a classic and well-studied problem.

The key advantage of transshipment over shortest paths is the so-called *boosting* property: one can often boost a crude approximate solution to a (near-optimal) $(1+\varepsilon)$-approximate solution. However, our understanding of this phenomenon is limited: it is not clear which approximators can be boosted. Moreover, all current boosting frameworks are built with a specific type of approximator in mind and are relatively complicated.

The main takeaway of our paper is conceptual: *any* black-box oracle that computes an approximate *dual* solution can be boosted to an $(1+\varepsilon)$-approximator. This decouples and simplifies all known near-optimal $(1+\varepsilon)$-approximate transshipment and shortest paths results: they all (implicitly) construct approximate dual solutions and boost them.

We provide a very simple analysis based on the multiplicative weights framework. Furthermore, to keep the paper completely self-contained, we provide a new (and arguably much simpler) analysis of multiplicative weights that leverages well-known optimization tools to bypass the ad-hoc calculations used in the standard analyses.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Theory of computation → Mathematical optimization; Theory of computation → Mixed discrete-continuous optimization; Theory of computation → Shortest paths; Theory of computation → Parallel algorithms

**Keywords and phrases** mixed continuous-discrete optimization, boosting, multiplicative weights, theoretical computer science, shortest path, parallel algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2023.104

**Related Version** *Full Version*: `https://arxiv.org/abs/2110.11723`

**Funding** *Goran Zuzic*: Work presented in this paper was partially performed while at ETH Zürich. Supported in part by NSF grants CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808, a Sloan Research Fellowship, and funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 949272).

**Acknowledgements** The author would like to thank Bernhard Haeupler, Patrik Pavic, and Richard Peng for helpful discussions about the paper. The author would also like to thank the anonymous reviewers for their helpful suggestions that significantly improved the quality of the paper.

## 1 Introduction

Suppose we are given a weighted graph $G=(V,E)$ and a *demand vector* $d \in \mathbb{R}^V$ satisfying $\sum_{v\in V} d(v) = 0$, where $d(v)$ denotes the number of units of some (single) commodity that are either available (if $d(v) > 0$) or required (if $d(v) < 0$) at the node $v$. *Transshipment* asks to distribute the available units of the commodity until it perfectly matches the requirement. The goal is to minimize the total cost of movement, where moving a single unit over an edge $e$ has a cost of $w(e)$ ($w(e) \geq 0$ is the weight of $e$).


Transshipment is a strong generalization of multiple problems, including the $s-t$ shortest path problem, optimal transport, or the assignment problem on metric spaces.

▶ **Example 1.** Given two nodes $s, t$ in a weighted graph $G$, the shortest path can be modeled as transshipment for the demand $d(v) := \mathbb{I}[v = s] - \mathbb{I}[v = t]$.

▶ **Example 2.** Let $(V, d)$ be a metric space and let $A, B \subseteq V$ be two disjoint subsets. The minimum cost perfect matching between $A$ and $B$ (the so-called assignment problem [14]) can be modeled as transshipment on the complete bipartite graph $(A \cup B, A \times B)$ with weights $w(\{a, b\}) = d(a, b)$ and the demand vector $d(v) := \mathbb{I}[v \in A] - \mathbb{I}[v \in B]$. This also models the Wasserstein distance (also known as the earth mover's distance or optimal transport [18]).

Perhaps surprisingly, transshipment has proven to be extremely useful for uncovering the distance structure (i.e., shortest paths) of a graph. Indeed, the problem has been the centerpiece of all near-optimal approaches to the single-source shortest path problem in the parallel and distributed settings [5, 10, 2, 19, 15].

The key property that differentiates transshipment from other similar problems like shortest path is the so-called *boosting* property – one can boost a crude, say $\text{poly}(\log n)$-approximate solution, to a near-optimal $(1 + \varepsilon)$-approximate solution. This conceptually reduces $(1+\varepsilon)$-transshipment (and shortest path computation) to approximate transshipment. However, not all approximators can be boosted and a more principled understanding of which approaches are susceptible to boosting is required. Moreover, the current boosting algorithms are coupled together with the specific approximators they use, making them non-modular, complicated, and hard to reuse.

The main takeaway of our paper is conceptual: *any* black-box oracle that computes a $\alpha$-approximate *dual* solution can be boosted to a $(1 + \varepsilon)$-approximate dual solution. This significantly simplifies current transshipment results by decoupling them into two independent questions: (1) how to obtain an approximate dual solution (which is often model-specific), and (2) how to boost this approximate solution (which can be reused). The scope of this paper is to develop a simple framework for the latter question.

We provide a very simple algorithm and analysis based on the *multiplicative weights framework*. Furthermore, to keep the paper completely self-contained, we provide a new (and arguably much simpler) analysis of multiplicative weights that leverages well-known optimization tools to bypass the ad-hoc calculations used in the standard analyses. (deferred to the full version).

We now provide several examples that show how prior approaches all (implicitly) construct approximate dual and then boost them.

▶ **Example 3.** Sherman [17] gave the first sequential almost-linear[1] $(1 + \varepsilon)$-transshipment algorithm. The main insight was the construction of a so-called *linear cost approximator*, which is a linear operator $R$ (i.e., matrix) such that $\|Rd\|_1$ approximates the optimal transshipment cost in the sense that $\text{OPT}(d) \leq \|Rd\|_1 \leq n^{o(1)}\text{OPT}(d)$ for all demands $d \in \mathbb{R}^V$. Their paper uses linear cost approximators with subgradient descent to show one can obtain a $(1 + \varepsilon)$-approximate solution. We provide a conceptual decoupling and reinterpretation of their paper: one can use any linear cost approximator $R$ to directly obtain an approximate dual solution, which can, in turn, be boosted to an $(1 + \varepsilon)$-approximate solution via our framework.

[1] We refer to $m\text{poly}(\log n)$ as near-linear and $m^{1+o(1)}$ as almost-linear runtimes.

▶ **Example 4.** Haeupler and Li [8] solve $n^{o(1)}$-approximate transshipment in the distributed setting and leave the possibility of boosting to an $(1+\varepsilon)$-approximation as the main open problem, which would have yielded important consequences in the distributed setting. Our paper provides a partial explanation to why their approach was not susceptible to boosting: their approach, based on low-stretch spanning trees, only computes a *primal* solution (i.e., an approximate flow), whereas a dual solution is required. A dual-based solution was later recently developed by Rozhon et al. [15].

▶ **Example 5.** Other successful $(1+\varepsilon)$-transshipment approaches either approximate the solution by solving the original problem on a spanner [5], or by constructing a linear cost approximator on an emulator of the original graph [10, 2] (an emulator of $G$ is a graph $H$ whose distance structure multiplicatively approximates the one of $G$, see Section 4.1). We show all of these approaches can be reinterpreted as obtaining an approximate dual solution.

**Comparison with Becker et al. [5].** The paper contributed the first polylog-competitive existentially-optimal shortest path algorithm in the distributed setting (up to $\tilde{O}(1)$-factors). Crucially, they develop a boosting framework for transshipment which, similar to this paper, uses an approximate dual solver to construct a near-optimal solution. The main drawbacks of their solver are that (1) the analysis of [5] is quite involved, stemming from it being based on projected gradient descent, and (2) as written, the interface of the [5] solver relies on solving a modified version of transshipment which is harder to interpret and work with than the original one. As stated in the journal version of [5], their interface can be significantly simplified (by working with projections), but this degrades the runtime to have an $\alpha^4$-dependency w.r.t. the approximation quality $\alpha$ of the approximator (we provide an $\alpha^2$-dependency) and requires non-explicit modifications to the solver that might be difficult for non-experts.[2] On the other hand, the approach presented in our paper has several drawbacks compared to [5], such as: (1) our solver requires a guess on the optimal solution, which is obtained using binary search, while their solver does not need adapting the internal parameters during the optimization process, and (2) our dual-only solver needs to perform extra steps to return a feasible primal solution. However, independent of the drawbacks, we believe the user-friendly interface, better runtime, and a simpler analysis make our conceptual contribution worthwhile.

**Potential impact.** Ultimately, we hope that this paper will encourage an ongoing effort to simplify deep algorithmic results that use continuous optimization tools. Such an effort would potentially yield a dual benefit: it would both lower the barrier to entry for newcomers by conceptually simplifying the current approaches, as well as help to transfer the modern theoretically-optimal algorithms into real-world state-of-the-art by allowing practitioners to independently combine the theoretical ideas with the many heuristics necessary for an algorithm to perform well in practice.

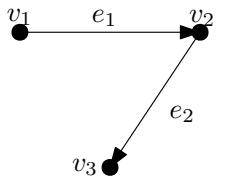
**Organization of the paper.** We present a model-oblivious boosting framework for transshipment in Section 3 and apply it in Section 4 to simplify previous results. These applications are loosely grouped by the method of computing the approximate dual solution: Section 4.1 presents results when the approximate solution is computed on a spanner or emulator (i.e.,

[2] Specifically, they require the returned dual solution be orthogonal to the demand vector. However, as they note in the journal version of their paper, this issue can be mitigated by projecting a general solution to the space of vectors orthogonal to the demand: this can be shown to work with a loss in approximation factor and time complexity if one appropriately initializes the solution.

on graphs that approximate the original metric). Section 4.2 presents results that compute the dual solution via (aforementioned) linear cost approximators. Finally, the full version of the paper gives a simple and self-contained analysis of multiplicative weights.

## 2 Preliminaries

**Graph Notation.** Let $G = (V, E)$ be a undirected graph and let $n := |V|, m := |E|$. It is often convenient to direct $E$ consistently. For simplicity and without loss of generality, we assume that $V = \{v_1, v_2, \ldots, v_n\}$ and define $\vec{E} = \{(v_i, v_j) \mid (v_i, v_j) \in E, i < j\}$. We identify $E$ and $\vec{E}$ by the obvious bijection. We chose this orientation for simplicity and concreteness: arbitrarily changing the orientations does not influence the results (if done consistently). We denote with $B \in \{-1, 0, 1\}^{V \times \vec{E}}$ the node-edge incidence matrix of $G$, which for any $v \in V$ and $e = (s, t) \in \vec{E}$ assigns $B_{s,e} = 1$, $B_{t,e} = -1$, and $B_{u,e} = 0$ when $u \notin \{s, t\}$. A weight or length function $w$ assigns each edge $e \in \vec{E}$ a weight $w(e) > 0$. The weight function can also be interpreted as a diagonal *weight matrix* $W \in \mathbb{R}^{\vec{E} \times \vec{E}}_{\geq 0}$ which assigns $W_{e,e} = w(e) \geq 1$ for any $e \in \vec{E}$ (and 0 on all off-diagonal entries). In this paper, it is often more convenient to specify weighted graphs via $G \cong (B, W)$, i.e., by specifying its matrices $B$ and $W$ as defined above.




$$B = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{cc} e_1 & e_2 \end{array} \\ \begin{bmatrix} +1 & 0 \\ -1 & +1 \\ 0 & -1 \end{bmatrix} \end{array}$$

**Figure 1** A simple graph $G$ and its corresponding node-edge incidence matrix $B$.

**Flows and Transshipment (TS).** A *demand* is a $d \in \mathbb{R}^V$. We say a demand is *proper* if $\sum_{v \in V} d_v = 0$. A *flow* is a vector $f \in \mathbb{R}^{\vec{E}}$, where $f(\vec{e}) > 0$ if flowing in the direction of the arc $\vec{e}$ and negative otherwise. A flow $f$ *routes* a demand $d$ if $Bf = d$. It is easy to see only proper demands are routed by flows. The cost of a flow $f$ is $\|Wf\|_1$. For a weighted graph $G$ and a given proper demand $d$ the *transshipment problem* (or TS, for short) asks to find a flow $f^*_d$ of minimum-cost among flows that route $d$. In other words, the tuple $(B, W, d)$ specifies a transshipment instance. When the underlying graph $G \cong (B, W)$ is clear from the context, we define $\|d\|_{\text{OPT}} := \|Wf^*_d\|_1$ to denote the cost of the optimal flow for routing demand $d$. The transshipment problem naturally admits the following LP formulation and its dual. The primal asks us to optimize over all *flows* $f \in \mathbb{R}^{\vec{E}}$, while the dual asks us to optimize over all vectors $\phi : \mathbb{R}^V$, which we refer to as *potentials*.

$$\textbf{Primal:} \quad \min \ \|Wf\|_1 : Bf = d, \qquad \textbf{Dual:} \quad \max \ \langle d, \phi \rangle : \left\|W^{-1}B^\top \phi\right\|_\infty \leq 1. \tag{1}$$

Scalar products are denoted as $\langle x, y \rangle = x^T \cdot y$. Finally, we assume the weights and demands are polynomially-bounded, hence $\|d\|_{\text{OPT}} \leq n^{O(1)}$. Any feasible primal and dual values provide an upper and lower bound on $\|d\|_{\text{OPT}}$, formally stated in the following well-known result.

▶ **Fact 6.** *Let $f \in \mathbb{R}^{\vec{E}}$ and $\phi \in \mathbb{R}^V$ be any feasible primal and dual solution: i.e., $Bf = d$ and $\left\|W^{-1}B^T\phi\right\|_\infty \leq 1$. Then $\langle d, \phi \rangle \leq \|d\|_{\text{OPT}} \leq \|Wf\|_1$.*

For example, consider the $s$-$t$ shortest path subproblem where $d(v) = \mathbb{I}[v = s] - \mathbb{I}[v = t]$ specifies the demand. One optimal solution to the primal/dual pair is to set $f(\vec{e})$ to 1 iff $\vec{e}$ is on some fixed shortest path from $s$ to $t$; $\phi(v)$ is set to the distance in $G$ from $t \in V$. Note that in this case the primal and dual objectives are equal, and correspond to the weight of the shortest path from $s$ to $t$.

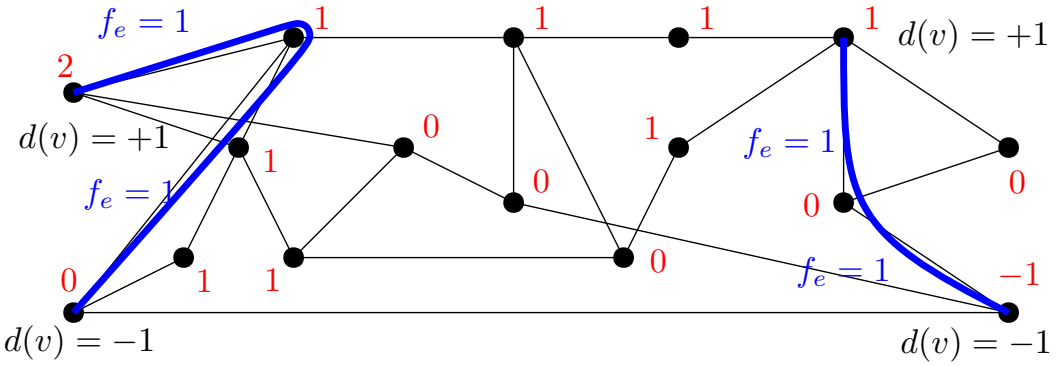



**Figure 2** A example transshipment graph with its exact solution. The original graph is unit weight $w_e = 1$ and undirected. The demand $d$ is non-zero at four nodes. The optimal primal flow $f$ is depicted in blue and is non-zero for four edges. One of many optimal vectors $\phi$ is depicted in red. The optimal value of the solution is OPT = 4.

**Asymptotic Notation.** We use $\tilde{O}$ to hide polylogarithmic factors in $n$, i.e., $\tilde{O}(1) = \text{polylog } n$.

**Algorithmic model and basic vector operations.** To facilitate both simplicity and generality, we specify our algorithms using high-level operations. Specifically, in a unit operation, we can perform the following so-called **basic vector operations**: (1) assign vectors in $\mathbb{R}^n$ or $\mathbb{R}^m$ to variables, (2) add two (vector) variables together, (3) apply any scalar function $\lambda : \mathbb{R} \to \mathbb{R}$ to each component of a vector separately, and (4) compute matrix-vector products with matrices $B$, $B^T$, $W$, and $W^{-1}$. Note that each basic vector operation can be near-optimally compiled into standard parallel/distributed models. In PRAM: each operation can be performed in $\tilde{O}(1)$ depth and near-linear work. In the standard distributed model of computation CONGEST [12] basic vector operations can be computed in a single round of distributed computation (where the variables are stored in the obvious distributed fashion).

**Multiplicative weights (MW) framework** is a powerful meta-algorithm that allows for (among other things) solving various optimization tasks by reducing them to simpler (so-called "linearized") versions of the original problem [3]. For the purposes of this paper, we define the following pair of tasks.

▶ **Definition 7.** *Let $\gamma \in \mathbb{R}$ be a scalar, $A \in \mathbb{R}^{m \times n}$ be a matrix, and $b \in \mathbb{R}^n$ be a vector. We define the following* ***Feasibility task****:*

$$\exists ? x \in \mathbb{R}^n \mid \quad \|Ax\|_\infty + \langle b, x \rangle \leq \gamma. \tag{2}$$

*Given additionally a vector $p \in \mathbb{R}^m$ satisfying $\|p\|_1 \leq 1$ and an accuracy parameter $\varepsilon > 0$, we also define the following* ***Linearized task****:*

$$\exists ? x \in \mathbb{R}^n \mid \quad \langle p, Ax \rangle + \langle b, x \rangle \leq \gamma - \varepsilon. \tag{3}$$

Note that if the Feasibility task (Equation (2)) is feasible, then the Linearized task (Equation (3)) is also feasible for every $p$ (satisfying $\|p\|_1 \leq 1$) and every $\varepsilon > 0$.

Suppose we want to solve some fixed Feasibility task (Equation (2)) and assume we know how to solve the accompanying (typically much easier!) Linearized task (Equation (3), for any $p$ and $\varepsilon$) via some *black-box Oracle*. Then, there exists a simple algorithm that computes a solution to the Feasibility task by repeatedly querying the Oracle with different values of $p$ that satisfy $\|p\|_1 \leq 1$ (the accuracy $\varepsilon$ stays fixed); the oracle is assumed to return a feasible solution $x$ for each queried Linearized task.

We define the **width of the Oracle** $\rho > 0$ to be (any upper bound on) the largest width of a solution $\|Ax\|_\infty$ that can be returned by the Oracle, i.e., $\rho \geq \|Ax\|_\infty$ during the course of the algorithm. Oracles with larger widths need to be queried more times, hence we aim to construct Oracles with their width being as small as possible. The following Theorem 8 and Algorithm 1 give a solver for the Feasibility task (Equation (2)) assuming the Oracle. We defer the proof to the full version of the paper.

▶ **Theorem 8.** *Let $(A, b, \gamma)$ be a feasible Feasibility task (Equation (2)) and fix $\varepsilon > 0$. Suppose we have access to an Oracle that will solve the accompanying Linearized task Equation (3) specified by $(A, b, \gamma, p, \varepsilon)$ for any $\|p\|_1 \leq 1$. Then, Algorithm 1 constructs a feasible solution for Equation (2) and queries the Oracle at most $4\varepsilon^{-2}\rho^2 \ln(2m)$ times. Here, $\rho > 0$ is the width of the Oracle.*

**Algorithm 1** Solver for the Feasibility task using an oracle for the Linearized task.

1. **Input:** Feasbility task $(A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^n, \gamma \in R)$ and $\varepsilon > 0$.
2. Initialize $x_* \leftarrow \vec{0} \in \mathbb{R}^n$ and $\beta := \varepsilon/(2\rho^2)$.
3. For $t = 1, \ldots, T$ rounds, where $T := 4\varepsilon^{-2}\rho^2 \ln(2m)$:
   a. Let $q \leftarrow \begin{bmatrix} A \\ -A \end{bmatrix} x_* \in \mathbb{R}^{2m}$.
   b. Let $q'_i \leftarrow \exp(\beta q_i)_i$ for $i \in [2m]$.
   c. Let $p_t \leftarrow (1/\sum_{i=1}^{2m} q'_i)(q'_i - q'_{i+m})$. (Normalization and flattening.)
   d. MW outputs $p_t \in \mathbb{R}^m$ to Oracle. (Note that $\|p_t\|_1 \leq 1$.)
   e. Oracle returns a solution $x_t \in \mathbb{R}^n$ to the Linearized task w.r.t. $p_t$. ($\rho$ must be set large enough such that $\|Ax\|_\infty \leq \rho$.)
   f. We update $x_* \leftarrow x_* + x_t$.
4. MW outputs $(1/T) \cdot x_* \in \mathbb{R}^n$.

## 3 A Boosting Framework for Transshipment

We describe how to compute an $(1+\varepsilon)$-approximate solution for transshipment given only a black-box oracle which computes an $\alpha$-approximate dual solution. This oracle is called the dual-only $\alpha$-approximator (where the dual is defined as in Equation (1)).

▶ **Definition 9** (Dual-Only Approximator). *Let $G \cong (B, W)$ be a weighted graph. A dual-only $\alpha$-approximator for transshipment over $G$ is an oracle which, given any proper demand $d \in \mathbb{R}^V$, outputs a dual solution $\phi \in \mathbb{R}^V$ satisfying the following properties:*

- *(Dual feasibility)* $\left\|W^{-1}B^T\phi\right\|_\infty \leq 1$.
- *(Approximation guarantee):* $\langle d, \phi\rangle \geq \frac{1}{\alpha}\|d\|_{\mathrm{OPT}}$ .

Note that, directly from its definition, a dual-only $\alpha$-approximator can be used to obtain an $\alpha$-approximate value of the solution.

We typically want poly$(\log n)$-approximators or $n^{o(1)}$-approximators that can be constructed and queried efficiently. However, for pedagogical purposes, we first show that the minimum spanning tree (MST) is a non-trivial $n$-approximator.

▶ **Example 10** (MST). Let $d \in \mathbb{R}^V$ be an arbitrary proper demand. The MST $T$ of $G$ can be used as a simple $n$-approximator for transshipment. First, root the MST $T$ in an arbitrary $r \in V$ and assume without loss of generality (up to re-orientation of edges in $E$) that all edges point from parent to child nodes in this rooted tree. Next, let $f_T$ be the unique flow supported on $T$ that perfectly routes $d$. We now define $\phi \in \mathbb{R}^V$ by saying $\phi(r) := 0$ and proceeding in a top-to-bottom order. For each parent-child tree-edge $e = (p, c)$, set $\phi(c) := \phi(p) - \text{sign}(f_T(e))w(e)$. Now, by construction, we have $\|Wf_T\|_1 = \langle \phi, d \rangle$ (decompose the flow $f_T$ into a positive combination of oriented paths such that all of them cross each tree edge with the same orientation; the claim is true for each one of them and, therefore, for their sum). Furthermore, $\left\|W^{-1}B^T\phi\right\|_\infty \leq n$ by the following argument: consider each edge $e = \{u, v\} \in E_G$ and consider the unique $u - v$ tree-path. This path is composed of at most $n$ edges, and each one of them have weight at most $w(e)$ (since $T$ is the MST). Hence, $|\phi(u) - \phi(v)| \leq w(e) \cdot n$, which is equivalent to the claim above. Finally, defining $\phi_* := \frac{1}{n}\phi$ we get a feasible dual solution that is $n$-approximate: Using Fact 6 we have $||d||_{\text{OPT}} \leq \|Wf_T\|_1 = \langle \phi, d \rangle = n \cdot \langle \phi_*, d \rangle$, as required.

We now show the central claim of our framework: given a dual-only $\alpha$-approximator we can leverage the multiplicative weights framework (Definition 7) to provide feasible potentials (i.e., a dual solution) $\phi \in \mathbb{R}^V$ that are $(1+\varepsilon)$-approximate, i.e., $\langle d, \phi \rangle \geq \frac{1}{1+\varepsilon} \|d\|_{\text{OPT}}$. The existence of the boosting procedure is formalized in Lemma 11, while the explicit algorithm is deferred to the full version of the paper.

As an assumption to simplify the exposition, we can safely assume that we know the value of $g := \|d\|_{\text{OPT}}$ (up to a multiplicative $1+\varepsilon$) as this value can be "guessed" via a standard binary search since our method will either certify that $\|d\|_{\text{OPT}} \leq (1+\varepsilon)g$ (telling us our guess $g$ is too low), or will otherwise construct a feasible solution $\phi$ with $\langle d, \phi \rangle \geq g$ (telling us we can increase our guess).

▶ **Lemma 11.** *Let $(B, W, d)$ be a transshipment instance and let $\varepsilon > 0$. Given any $g \geq 0$, and any dual-only $\alpha$-approximator, there is a $4\varepsilon^{-2}\alpha^2 \ln(2m)$-round algorithm that, in each round, queries the approximator once and performs $O(1)$ basic vector operations. At termination, the algorithm either:*

1. *outputs (feasible) potentials $\phi_* \in \mathbb{R}^V$ satisfying $\left\|W^{-1}B^T\phi_*\right\|_\infty \leq 1$ and $\langle d, \phi_* \rangle \geq g$, or,*
2. *detects that $\|d\|_{\text{OPT}} \leq (1+\varepsilon)g$; indeed, it outputs an (infeasible) flow $f_* \in \mathbb{R}^E$ satisfying $\|Wf_*\|_1 \leq g$ and $\|d - Bf_*\|_{\text{OPT}} \leq \varepsilon g$.*

**Remark.** If one is only concerned about finding an $(1+\varepsilon)$-approximate dual solution, one can completely ignore the infeasible flow that is being outputted and simply use the fact that the second condition guarantees $\|d\|_{\text{OPT}} \leq (1+\varepsilon)g$, which is sufficient for binary search. Furthermore, we note that any such (infeasible) flow $f_*$ satisfying the above properties guarantees $\|d\|_{\text{OPT}} \leq (1+\varepsilon)g$ by the following argument. First, by definition of $\|d - Bf_*\|_{\text{OPT}} \leq \varepsilon g$, there exists a flow $f_{\text{residual}}$ that routes $d - Bf_*$ and has cost $\|Wf_{\text{residual}}\|_1 \leq \varepsilon g$. Then, the flow $f_* + f_{\text{residual}}$ routes demand $d$ (since $Bf_* + Bf_{\text{residual}} = Bf_* + d - Bf_* = d$) and has cost at most $\|Wf_*\|_1 + \|Wf_{\text{residual}}\|_1 \leq g + \varepsilon g = (1+\varepsilon)g$, implying that $\|d\|_{\text{OPT}} \leq (1+\varepsilon)g$.

**Proof.** First, finding potentials $\phi \in \mathbb{R}^V$ satisfying $\left\|W^{-1}B^T\phi\right\|_\infty \leq 1$ and $\langle d, \phi\rangle \geq g$ is equivalent to finding potentials $\exists ?\phi \in \mathbb{R}^V \mid \left\|W^{-1}B^T\phi\right\|_\infty - \left\langle \frac{1}{g}d, \phi\right\rangle \leq 0$ (one direction is immediate, the other direction follows by the fact that we can scale $\phi$ such that $\langle d, \phi\rangle = g$). Therefore, it is sufficient to solve the following so-called TS Feasibility task (see Figure 3).

| | | |
|---|---|---|
| **TS feasbility task**: | $\exists ?\phi \in \mathbb{R}^V \mid$ | $\left\|W^{-1}B^T\phi\right\|_\infty \leq 1$ and $\langle d, \phi\rangle \geq g$. |
| **TS feasbility task** (equivalent): | $\exists ?\phi \in \mathbb{R}^V \mid$ | $\left\|W^{-1}B^T\phi\right\|_\infty - \left\langle \frac{1}{g}d, \phi\right\rangle \leq 0$. |
| **Linearized TS task** (given $\|p\|_1 \leq 1$): | $\exists ?\phi \in \mathbb{R}^V \mid$ | $\left\langle p, W^{-1}B^T\phi\right\rangle - \left\langle \frac{1}{g}d, \phi\right\rangle \leq -\varepsilon$. |
| **Linearized TS task** (equivalent): | $\exists ?\phi \in \mathbb{R}^V \mid$ | $\langle d_{\text{residual}}, \phi\rangle \geq \varepsilon \cdot g$, where $d_{\text{residual}} := d - B(g \cdot W^{-1}p)$. |

**Figure 3** The (second, equivalent form of the) TS Feasibility task is a subcase of the Feasibility task defined in Definition 7 with $A := W^{-1}B^T, b := (1/g)d, \gamma := 0$, and renaming $x \to \phi$. The equivalent forms of the tasks follow by straightforward algebraic manipulation.

We apply the MW framework by interpreting the TS Feasbility task as a Feasibility Task in the sense of Definition 7, solving it via Algorithm 1 where we have to implement the Oracle.

First, we note that the TS Feasibility task directly corresponds to a $(A := W^{-1}B^T, b := -\frac{1}{g}d, \gamma := 0)$-Feasibility task. We aim to implement the Oracle for the corresponding Linearized TS task with a small width $\rho$. To recap, $\rho$ is the maximum value of $\left\|W^{-1}B^T\phi\right\|_\infty$ ever returned by the Oracle – we later determine that setting $\rho := \alpha$ suffices.

The Oracle, upon receiving $p$ by Algorithm 1, queries the dual-only $\alpha$-approximator with the (so-called) residual demand $d_{\text{residual}} := d - B(g \cdot W^{-1}p)$. Intuitively, we can interpret $p$, or more specifically $g \cdot W^{-1}p$, as the "current" iterate of the final flow solution. Specifically, $\left\|g \cdot W^{-1}p\right\|_1 \leq g$, i.e., it has a small cost since $g$ is a guess for OPT. If the residual demand can be routed with a small cost of at most $\varepsilon g$ (which can be estimated via the approximator), it means that $\|d_{\text{residual}}\| \leq \varepsilon g$, hence $f_* := g \cdot W^{-1}p$ satisfies the second output condition.

The appoximator, being asked to route $d_{\text{residual}}$, returns the $\alpha$-approximate feasible dual, i.e., a vector $\phi_{\text{residual}} \in \mathbb{R}^V$ satisfying $\langle d_{\text{residual}}, \phi_{\text{residual}}\rangle \geq \frac{1}{\alpha}\|d_{\text{residual}}\|_{\text{OPT}}$ and $\left\|W^{-1}B^T\phi_{\text{residual}}\right\|_\infty \leq 1$. The Oracle outputs $\phi' := \alpha \cdot \phi_{\text{residual}}$. Note that the width of the oracle is exactly $\left\|W^{-1}B\phi'\right\|_\infty = |\alpha| \left\|W^{-1}B\phi_{\text{residual}}\right\|_\infty \leq \alpha \cdot 1 = \alpha$.

Either $\langle d_{\text{residual}}, \phi'\rangle \geq \varepsilon \cdot g$, and the Oracle successfully solves the Linearized TS task by returning $\phi_{\text{residual}}$, in which case the MW loop continues. If this is always the case, Algorithm 1 outputs $\phi_*$ satisfying $\left\|W^{-1}B^T\phi_*\right\|_\infty \leq 1$ and $\langle d, \phi_*\rangle \geq g$, as required. Regarding the width of the solution, we have that $\left\|W^{-1}B^T\phi'\right\|_\infty \leq \alpha$, hence setting $\rho := \alpha$ suffices, leading to at most $4\varepsilon^{-2}\alpha^2 \ln 2m$ rounds of the algorithm.

On the other hand, if this is (ever) not the case, we say the Oracle fails. In this case, at the moment of failure, we define $f_* := g \cdot W^{-1}p$ and have that $\langle d_{\text{residual}}, \phi'\rangle \leq \varepsilon \cdot g$. Since $\phi' = \alpha\phi_{\text{residual}}$, we have $\langle d_{\text{residual}}, \phi_{\text{residual}}\rangle \leq \frac{\varepsilon}{\alpha} \cdot g$. Since $\phi_{\text{residual}}$ is an $\alpha$-approximate dual w.r.t. $d_{\text{residual}}$, we have that $\left\|d - B(g \cdot W^{-1}p)\right\|_{\text{OPT}} = \|d_{\text{residual}}\|_{\text{OPT}} \leq \alpha \cdot \frac{\varepsilon}{\alpha}g = \varepsilon \cdot g$. Therefore, $f_*$ satisfies the second condition and we are done. ◀

The full algorithm is deferred to the full version.

**Reducing the residual error of the primal.** While the booster of Lemma 11 returns a feasible $(1+\varepsilon)$-approximate dual solution, it does not return a feasible primal solution (i.e., flow). However, this issue can be resolved by repeatedly routing the residual demand $d - Bf_*$

until the cost of routing the residual demand drops to an insignificant $1/\text{poly}(n)$-fraction of the original cost, at which point any trivial reparation scheme suffices (like routing along the MST). See the full version of the paper for more details. Combining the above result with binary searching the guess $g$ and with the residual error reduction (but without the model-specific trivial routing), we immediately yield the following result (proof deferred to the full version).

▶ **Corollary 12.** *Let $(B, W, d)$ be a transshipment instance. Given any $1/2 \geq \varepsilon > 0$, $C > 0$ and dual-only $\alpha$-approximator, there is an $\tilde{O}(C \cdot \varepsilon^{-2}\alpha^2)$-round algorithm computing (both):*

- *a feasible dual $\phi_*$ satisfying $(1+\varepsilon)^{-1} \|d\|_{\text{OPT}} \leq \langle d, \phi \rangle \leq \|d\|_{\text{OPT}}$, and,*
- *an (infeasible) primal $f_*$ satisfying $\|Wf\|_1 \leq (1+\varepsilon)\|d\|_{\text{OPT}}$ and $\|d - Bf_*\|_{\text{OPT}} \leq n^{-C}\|d\|_{\text{OPT}}$.*

*In each round, the algorithm performs $O(1)$ basic vector operations and queries to the approximator.*

## 4 Applications

In this section, we show how to apply the boosting framework of Section 3 to simplify and decouple several landmark results in the parallel and distributed settings. First, we describe results that approximate transshipment by solving it on a compact graph representation called a spanner or emulator (Section 4.1). Then, we describe results that use linear cost approximators (Section 4.2).

### 4.1 Approximating via spanners and emulators

A $\beta$-approximate **emulator** of a graph $G = (V, E_G)$ is a weighted graph $H = (V, E_H)$ on the same vertex set where the distances are approximated with a distortion of $\beta$; i.e., $\text{dist}_G(u,v) \leq \text{dist}_H(u,v) \leq \beta \cdot \text{dist}_G(u,v)$ for all $u, v \in V$. A spanner is simply an emulator that is a subgraph of $G$, i.e., where $E_H \subseteq E_G$, making it particularly well-studied in some settings.

Approximating with emulators is conceptually straightforward: faced with a transshipment instance on $G$, we (approximately) solve the instance on $H$, which yields an approximate solution on $G$. This is captured by the following result.

▶ **Theorem 13.** *Let $H$ be a $\beta$-approximate emulator of $G$. Any dual-only $\alpha$-approximator on $H$ is a dual-only $(\alpha \cdot \beta)$-approximator on $H$.*

**Proof.** Fix a demand $d$ on $G$. Querying the $H$-approximator, we obtain a dual solution $\phi_H$ satisfying $\left\|W_H^{-1} B_H^T \phi_H\right\|_\infty \leq 1$; we also know an accompanying primal solution $f_H$ exists with $W_H(f_H) \leq \alpha \langle d, \phi_H \rangle$.

*Primal solution.* We construct a flow $f_G$ in $G$ as follows. For each edge $e \in E_H$ we know, due to $\text{dist}_G(u,v) \leq \text{dist}_H(u,v)$, that there exists a path in $G$ of length at most $w_H(e)$; we add $f_H(e)$ amount of flow along this path. It is easy to check that, $f_G$ routes $d$ (i.e., $B_G f_G = d$, hence it is feasible) and that $W_G(f_G) \leq W_H(f_H)$, hence $\|d\|_{\text{OPT}(G)} \leq \alpha \langle d, \phi_H \rangle$.

*Dual solution.* Let $\phi_G := \frac{1}{\beta}\phi_H$. Note that $\|d\|_{\text{OPT}(G)} \leq (\alpha\beta) \cdot \langle d, \phi_G \rangle$, hence it is sufficient to deduce $\left\|W_G^{-1} B_G^T \phi_G\right\|_\infty \leq 1$. Since $\phi_H$ is feasible in $H$, we have for each $e' = \{u', v'\} \in E_H$ that $(B_H^T \phi_G)_{e'} = |\phi_G(u') - \phi_G(v')| \leq \frac{|\phi_H(u') - \phi_H(v')|}{\beta} = \frac{w_H(u',v')}{\beta}$. Fix an edge $e = \{u, v\} \in E_G$; since $\text{dist}_H(u,v) \leq \beta \cdot \text{dist}_G(u,v)$ there exists a path

$(u = p'_0, p'_1, p'_2, \ldots, p'_k = v)$ in $H$ of length at most $\beta \cdot w_G(e)$. Therefore, we can deduce that $\left\|W_G^{-1} B_G^T \phi_G\right\|_\infty \leq 1$ in the following way: $|(W_G^{-1} B_G^T \phi_G)_e| = \frac{|\phi_G(u) - \phi_G(v)|}{w_G(e)} \leq \frac{\sum_{i=1}^{T} |\phi_G(p'_{i-1}) - \phi_G(p'_i)|}{w_G(e)} \leq \frac{\sum_{i=1}^{T} w_H(p'_{i-1}, p'_i)}{\beta w_G(e)} \leq \frac{\beta w_G(e)}{\beta w_G(e)} = 1.$ ◀

**Remark.** There are a few immediate extensions to the above proof. Given a primal-dual approximator (one that returns both a primal and a dual) on a *spanner*, we can immediately obtain a primal-dual approximator on $G$ since the returned primal $f_H$ is also a feasible primal in $G$. A similar property holds for emulators, but one would need to provide a mapping that embeds each edge $e \in E_G$ into (paths of) $H$ that are of length at most $\beta \cdot w(e)$ in order to construct the flow $f_G$ on $G$.

**Application: TS in Broadcast congested clique [5].** Using algorithms from prior work, a Broadcast congested clique can compute an $\tilde{O}(1)$-approximate Baswana-Sen [4] spanner $H$ in $\tilde{O}(1)$ rounds. The edges of such a spanner are naturally partitioned into $n$ parts of size $\tilde{O}(1)$, where each part is associated with a unique node, and that node knows the edges in its part. Therefore, the spanner can be made global knowledge in $\tilde{O}(1)$ rounds using broadcasts. Therefore, each node can solve a transshipment instance on $H$, providing an $\tilde{O}(1)$-approximator for the original graph via Theorem 13, culminating in an $\tilde{O}(\varepsilon^{-2})$-round solution for $(1+\varepsilon)$-transshipment.

**Application: existentially-optimal SSSP in Broadcast CONGEST [5].** Consider the single-source shortest path (SSSP) problem where each node wants to compute $(1+\varepsilon)$-approximate from some source $s \in V$. From prior work, we can compute an *overlay graph* $G' = (V', E')$ where $V' \subseteq V$ and $|V'| = \tilde{O}(\varepsilon^{-1}\sqrt{n})$ such that the SSSP task on $G$ reduces to SSSP on $G'$, and $G'$ can be computed in $\tilde{O}(D + \varepsilon^{-1}\sqrt{n})$ rounds. As was shown in [5], an SSSP instance can be solved by solving $\tilde{O}(1)$ transshipment instances (the details are non-trivial and out of scope of this paper), hence the problem reduces to solving TS on $G'$. However, any $T$-round Broadcast congested clique algorithm can be simulated on $G'$ in $T \cdot O(D + |V'|) = T \cdot \tilde{O}(D + \varepsilon^{-1}\sqrt{n})$ rounds of Broadcast CONGEST: we simulate a single round by constructing a BFS tree on $G$ (of depth $O(D)$ and in $O(D)$ rounds), and then pipelining all $|V'|$ messages (that are to be broadcasted in the current round) to the root and them down to all other nodes, taking $O(D + |V'|)$ rounds in Broadcast CONGEST per round of Broadcast congested clique. Combining with the Broadcast congested clique result, we obtain an $\tilde{O}(\varepsilon^3)(D + \sqrt{n})$-round algorithm.

**Application: near-optimal TS in PRAM [2].** The paper introduces a concept called *low-hop emulator* $H = (V, E_H)$ of $G = (V, E)$ satisfying (i) $H$ is an $\tilde{O}(1)$-approximate emulator of $G$, (ii) $|E_H| = \tilde{O}(n)$, and (iii) $\text{dist}_H^{O(\log\log n)}(u, v) = \text{dist}_H(u, v)$, i.e., every (exact) shortest path in $H$ has at most $O(\log\log n)$ hops (edges). Moreover, low-hop emulators can be computed in PRAM in $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work. Low hop emulators are particularly useful since Property (iii) implies that one can compute (exact) SSSP on them in $\tilde{O}(1)$ depth and $\tilde{O}(n)$ work (e.g., using $O(\log\log n)$ rounds of Bellman-Ford). The ability to compute exact SSSP enables each node of $H$ to be embedded into $\ell_1$ space of dimension $\tilde{O}(1)$ with (worst-case) distortion $\tilde{O}(1)$ (via so-called Bourgain's embedding [6] via $\tilde{O}(1)$ SSSP oracle calls). Since $H$ is an emulator of $G$, the same embedding is an $\tilde{O}(1)$-distortion embedding of $G$. Using Theorem 13, this reduces $(1+\varepsilon)$-TS to finding an $\tilde{O}(1)$-approximator in $\ell_1$ space. This can be done in $\tilde{O}(1)$ depth and $\tilde{O}(n)$ work using linear cost approximators (explained in Section 4.2) by utilizing the so-called *randomly shifted grids* method [9]. This culminates in an $\tilde{O}(\varepsilon^{-2})$ depth and $\tilde{O}(\varepsilon^{-2}m)$ work $(1+\varepsilon)$-transshipment algorithm.

## 4.2 Approximating by linear cost approximators

A particularly successful type of approximator for transshipment has been the linear cost approximator. The successes of such an approximator include the first $m^{1+o(1)}$ algorithm for transshipment in the centralized model [17] and the first $\tilde{O}(m)$-work and $\tilde{O}(1)$-depth parallel shortest path algorithm [2, 10].

▶ **Definition 14.** *An $\alpha$-approximate linear cost approximator for a weighted graph $G$ is a $k \times n$ matrix $P$, such that, for any proper demand $d$ it holds that $\|d\|_{\text{OPT}} \leq \|Pd\|_1 \leq \alpha \|d\|_{\text{OPT}}$.*

Our insight is that one can immediately convert a linear cost approximator $P$ to a dual-only approximator. Note that the sign function is applied entry-wise to a vector.

▶ **Theorem 15.** *Let $P$ be an $\alpha$-approximate linear cost approximator. Consider the function $\phi(d)$ that maps a demand $d$ to $\phi(d) := \frac{1}{\alpha} P^T \text{sign}(Pd)$. Then, $\phi$ is a dual-only $\alpha$-approximator.*

**Proof.** Let $G \cong (B, W)$ be the underlying graph. First, we show that the following subclaim about a linear-algebraic guarantee that characterizes $P$: we have that $\left\|yPBW^{-1}\right\|_\infty \leq \alpha$ over all $\|y\|_\infty \leq 1$. Specifically, for each oriented edge $\vec{e} \in \vec{E}$, consider how $P$ approximates the cost of routing a unit from the head to the tail of $\vec{e}$. Formally, we define the demand $d_{\vec{e}}$ to be $d_{\vec{e}}(x) := \mathbb{I}[x = s] - \mathbb{I}[x = t]$ for an edge $\vec{e} = (s, t) \in \vec{E}$. Clearly, $\|d_{\vec{e}}\|_{\text{OPT}} \leq w(e)$, hence it is necessary that $\left\|Pd_{\vec{e}} w(e)^{-1}\right\|_1 \leq \alpha$. Furthermore, it is easy to see that the columns of $B$ are exactly $d_{\vec{e}}$ over all $\vec{e} \in \vec{E}$, hence each column of $PBW^{-1}$ has $\ell_1$-norm at most $\alpha$. This is equivalent to $\left\|y^T PBW^{-1}\right\|_\infty \leq \alpha$ over all $\|y\|_\infty \leq 1$. This proves the subclaim.

We now prove the complete result. Let $y := \text{sign}(Pd)$ and $\phi(d) := \frac{1}{\alpha} P^T y$. Since, $\|d\|_{\text{OPT}} \leq \|Pd\|_1$, there must exists a flow $f$ satisfying $d$ such that $\|d\|_{\text{OPT}} \leq \|Wf\|_1 \leq \|Pd\|_1$. We verify all properties Definition 9. (Dual feasibility) $\left\|W^{-1}B^T\phi(d)\right\|_\infty = \frac{1}{\alpha}\left\|W^{-1}B^T P^T y\right\|_\infty \leq \frac{1}{\alpha} \cdot \alpha = 1$ via the subclaim. (Approximation guarantee) $\langle d, \phi(d)\rangle = \frac{1}{\alpha}\langle Pd, y\rangle = \frac{1}{\alpha}\langle Pd, \text{sign}(Pd)\rangle = \frac{1}{\alpha}\|Pd\|_1 \geq \frac{1}{\alpha}\|d\|_{\text{OPT}}$. ◀

Having a dual-only $\alpha$-approximator that can be evaluated in $M$ time, we construct (via Corollary 12) an $\tilde{O}(\varepsilon^{-2}\alpha^2 \cdot M)$ time $(1+\varepsilon)$-approximate algorithm for transshipment.

▶ **Corollary 16.** *Let $P$ be an $\alpha$-approximate linear cost approximator on a weighted graph $G$ and suppose that we can evaluate matrix-vector multiplications with $P$ and $P^T$ (and other basic vector operations) in $M$ time. Given any TS instance, there is an $\tilde{O}(\varepsilon^{-2}\alpha^2 M)$-time algorithm that computes a $(1+\varepsilon)$-approximate primal-dual pair $(f, \phi)$ satisfying the properties listed in Corollary 12.*

**Application: almost-optimal sequential TS [17].** The goal is to construct $\varepsilon^{-2}m^{1+o(1)}$-time $(1+\varepsilon)$-TS solver in the sequential setting. Following Corollary 16, it is sufficient to construct a $n^{o(1)}$-approximate linear cost approximator $P$, which is accomplished as follows. Each vertex of a weighted graph $G$ is embedded into $\ell_1$ space of dimension $O(\log^2 n)$ with (worst-case) distortion $O(\log n)$ (via so-called Bourgain's embedding [6] in $\tilde{O}(m)$ sequential time). Then, the dimension of the embedding is reduced to $d := O(\sqrt{\log n})$ via a simple Johnson-Lindenstrauss projection [7], increasing the distortion of the embedding to $\exp(O(d)) = n^{o(1)}$. Finally, the paper constructs a $O(\log^{1.5} n)$-approximate linear cost approximator in this (virtual) $\ell_1$ space of dimension $d$ that can be evaluated efficiently, leading to a $\exp(O(d)) \cdot O(\log^{1.5} n) = n^{o(1)}$-approximate linear cost approximator in $G$, which yields the result. **Approximator in $\ell_1$ space:** We give a short cursory description on how to construct the approximator $P$. Re-scale and round the $\ell_1$ space such that all coordinates are integral.

Then, each point $x$ calculates the distance $c(x)$ to the closest point with all-even coordinates. Then, $x$ uniformly spreads its demand $d(x)$ among all points with all-even coordinates that are of distance exactly $c(x)$ to $x$. Finally, repeat the algorithm on points with all-even coordinates (delete other points, divide all coordinates by 2). After $O(\log n)$ iterations, the entire remaining demand will be supported on $2^d$ vertices of the hypercube, which can be routed to a common vertex yielding a $O(d)$ approximation. It can be shown that the cost incurred by spreading the demand at any particular step $O(d)$-approximates the optimal solution, and that the optimal solution does not increase in-between two steps, leading to a $O(d \log n) = O(\log^{1.5} n)$-approximate linear cost approximator. **Efficiency:** Evaluating the approximator requires computing the demands at each step in the above algorithm. Evaluating even the first step requires $n2^d$ time since each point $x$ sends its demand to (potentially) $2^d = n^{o(1)}$ closest all-even points. Therefore, the dimension of the embedding is reduced to $O(\sqrt{\log n})$. Moreover, the paper (implicitly) claims this approximator in $\ell_1$ can be evaluated in $m^{1+o(1)}$ time. Finally, we remark that the approximator does not yield a flow in the original graph in any meaningful way, (i.e., it only approximates costs), confirming that it is dual-only. Together, we solve $(1+\varepsilon)$-TS in $\varepsilon^{-2}m^{1+o(1)}$ time.

**Application: near-optimal TS in PRAM [10].** The goal is to solve $(1+\varepsilon)$-TS in $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work in PRAM. The paper constructs an $\tilde{O}(1)$-approximate linear cost approximator $P$ with sparsity $\tilde{O}(m)$, meaning it can be evaluated in $\tilde{O}(1)$ depth and $\tilde{O}(m)$ work, which would yield the result. To do so, the paper follows [17] by embedding $G$ in $\ell_1$ space with distortion $\tilde{O}(1)$ and dimension $d := \tilde{O}(1)$ and then uses the randomly shifted grids methods of [9] to approximate the cost in this virtual space. **Approximator in $\ell_1$ space:** We define a *randomly shifted grid* of scale $W$ to be the set $W(\mathbb{Z}^d + u) \subseteq \mathbb{R}^d$, where each coordinate of $u \in \mathbb{R}^d$ is uniformly drawn from $[0, 1)$ (i.e., one obtains a randomly shifted grid by taking all integral $d$-dimensional points, randomly translating them along each axis, them multiplying all coordinates by $W$). Initially, set $W \leftarrow \tilde{O}(1)$. The routing works by sampling $s := \tilde{O}(1)$ randomly shifted grids of scale $W$ and, for each grid, each point $x$ sends $1/s$ of its demand $d(x)$ to the closest point in the grid. The scale $W$ is increased by a polylogarithmic factor and the algorithm is repeated for $O(\log n)$ steps until all demand is supported on a hypercube, at which point it can be $O(d)$-approximated by aggregating it at a single vertex. It can be shown that the cost incurred by routing the demand at any particular step $\tilde{O}(1)$-approximates the optimal solution, and that the optimal solution increases only by a multiplicative $1 + 1/\text{poly}(\log n)$ factor, hence after $O(\log n)$ iterations we obtain an $\tilde{O}(1)$-approximate linear cost approximator $P$ that has sparsity $\tilde{O}(m)$. **Vertex reduction framework:** On its face, the above approach simply shows that in order to get $(1+\varepsilon)$-transshipment (and $(1+\varepsilon)$-shortest paths, as arduously shown in the paper), it is sufficient to find an $\tilde{O}(1)$-distortion $\ell_1$-embedding. However, to find an $\ell_1$-embedding, one needs $\tilde{O}(1)$-approx shortest paths (with some additional technical requirements concerning the violation of the triangle inequality). To resolve this cycle, the paper goes through the vertex reduction framework of [11, 13] which reduces the number of vertices by a polylogarithmic factor, recursively solves transshipment, lifts the solution to the original graph, and repairs it using the boosting framework. The details are out-of-scope.

**Future work.** The ideas used for solving transshipment have historically paralleled the ideas used for solving maximum flow problems. Adding to the connection between these two problems, approximate solutions to maximum flow can also be boosted in a similar way to transshipment [16] via linear cost approximators (called *congestion approximators*). However,

no framework that can handle black-box approximators has been developed – creating such a framework would conceptually simplify the task of designing approximate maximum flow solutions. Furthermore, both transshipment and maximum flow are special cases of the so-called $\ell_p$-norm flow, which also seems to support boosting [1].

## References


1 Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for $\ell_p$-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1405–1424. SIAM, 2019.

2 Alexandr Andoni, Clifford Stein, and Peilin Zhong. Parallel approximate undirected shortest paths via low hop emulators. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 322–335. ACM, 2020.

3 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.*, 8(1):121–164, 2012.

4 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

5 Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-Optimal Approximate Shortest Paths and Transshipment in Distributed and Streaming Models. In *31st International Symposium on Distributed Computing (DISC)*, volume 91, pages 7:1–7:16, 2017.

6 Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

7 Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the johnson-lindenstrauss lemma. *International Computer Science Institute, Technical Report*, 22(1):1–5, 1999.

8 Bernhard Haeupler and Jason Li. Faster distributed shortest path approximations via shortcuts. *arXiv preprint*, 2018. `arXiv:1802.03671`.

9 Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *3rd international workshop on statistical and computational theories of vision*, volume 2(3), page 5, 2003.

10 Jason Li. Faster parallel algorithm for approximate shortest path. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 308–321. ACM, 2020.

11 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 245–254, 2010.

12 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

13 Richard Peng. Approximate undirected maximum flows in $O(mpolylog(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1862–1867, 2016.

14 Lyle Ramshaw and Robert E Tarjan. On minimum-cost assignments in unbalanced bipartite graphs. *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1*, 20, 2012.

15 Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected (1+epsilon)-shortest paths via minor-aggregates: Near-optimal deterministic parallel & distributed algorithms. In *Proceedings of the 54rd Annual ACM Symposium on Theory of Computing (STOC)*, 2022.

16 Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 263–269, 2013.

**17** Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 772–780, 2017.

**18** Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

**19** Goran Zuzic, Goramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based l1-oblivious routing. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2022.